

**UNIVERSIDADE REGIONAL DE BLUMENAU**  
**CENTRO DE CIÊNCIAS EXATAS E NATURAIS**  
**CURSO DE CIÊNCIAS DA COMPUTAÇÃO**  
(Bacharelado)

**UTILIZAÇÃO DA TECNOLOGIA *ACTIVEX DATA*  
*OBJECTS(ADO)* EM UM SISTEMA COM OBJETOS  
DISTRIBUÍDOS**

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE  
REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA  
DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA  
COMPUTAÇÃO — BACHARELADO

**SUZETE TERESINHA COLLING**

BLUMENAU, JUNHO/2000

2000/1-64

# **UTILIZAÇÃO DA TECNOLOGIA *ACTIVEX DATA OBJECTS*(ADO) EM UM SISTEMA COM OBJETOS DISTRIBUÍDOS**

**SUZETE TERESINHA COLLING**

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

**BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO**

---

Prof. Maurício Capobianco Lopes — Orientador na FURB

---

Prof. José Roque Voltolini da Silva — Coordenador do TCC

## **BANCA EXAMINADORA**

---

Prof. Maurício Capobianco Lopes

---

Prof. Marcel Hugo

---

Prof. Wilson Pedro Carli

A meus pais, irmãos e meu noivo,  
pelo intenso apoio que sempre me deram.

# AGRADECIMENTOS

Ao meu orientador, Professor Maurício Capobianco Lopes. A orientação deste meu Mestre foi primordial para meu aprendizado! Agradeço por ter tido o privilégio de ter sido sua orientanda.

Agradeço, principalmente, aos meus pais, Roque e Maria Mercêdes, que sempre lutaram para proporcionar aos filhos tudo o que eles jamais tiveram para si. Sem o incentivo deles, eu jamais teria concluído o curso.

Aos meus irmãos, Neuza e Wanderlei, que mesmo distantes fisicamente, sempre estiveram apoiando e incentivando.

Ao meu noivo, Everton, que esteve constantemente do meu lado, compartilhando as horas felizes e também as tristes. Agradeço principalmente pela sua compreensão nos momentos em que mais precisei. Tivemos muitos momentos felizes, e também momentos difíceis: as dificuldades que passamos desde o início da faculdade, a saudade das pessoas queridas, as quedas que tivemos ao longo do caminho... Mas ao lembrarmos de tudo isso, também lembramos que estivemos sempre juntos, num apoio mútuo e constante. Obrigada, Everton, sua presença foi fundamental para que eu pudesse concluir este trabalho!

A Deus, pela sua constante presença em tudo o que fazemos. A fé é o impulso para lutarmos sempre, buscarmos sempre novas conquistas e quando elas acontecem, Deus está presente.

A todos os meus colegas de faculdade, que ajudaram para tornar este período de estudos estimulante e alegre.

Aos meus colegas de trabalho, que no dia-a-dia, contribuíram para o meu crescimento. Em especial àqueles que me auxiliaram durante a elaboração do TCC.

E a todos os outros amigos e parentes que incentivaram o meu estudo.

# Sumário

Lista de Figuras .....	vii
Lista de Tabelas .....	ix
Lista de Quadros .....	x
Lista de Abreviaturas .....	xi
RESUMO .....	xiii
ABSTRACT .....	xiv
1 INTRODUÇÃO .....	1
1.1 OBJETIVOS .....	3
1.2 ESTRUTURA .....	3
2 <i>ACTIVEX</i> DATA OBJECTS (ADO) .....	5
2.1 HISTÓRICO .....	5
2.2 DEFINIÇÕES .....	7
2.3 CONCEITO .....	8
2.4 UDA - UNIVERSAL DATA ACCESS .....	10
2.4.1 ADO E UDA .....	13
2.5 OLE DB .....	13
2.5.1 ADO E OLE DB .....	16
2.6 ADO E OLE DB COMO INTEGRANTES DE UDA MULTI PLATAFORMA .....	17
2.7 ARQUITETURAS DE CAMADAS .....	18
2.8 MODELO BÁSICO DE PROGRAMAÇÃO ADO .....	23
2.9 APLICAÇÕES DISTRIBUÍDAS COM ADO .....	26
2.9.1 OBJETOS DISTRIBUÍDOS .....	26
2.9.2 <i>ACTIVEX</i> .....	30
3 DELPHI 5.0 - INOVAÇÕES DO AMBIENTE .....	32

3.1 BANCO DE DADOS NO DELPHI 5.0.....	34
3.2 DELPHI 5.0 E ADO.....	35
4 DESENVOLVIMENTO DO PROTÓTIPO .....	36
4.1 O TUTOR INTELIGENTE PARA O AMBIENTE DELPHI .....	36
4.2 ESPECIFICAÇÃO DO PROTÓTIPO .....	37
4.2.1 CASOS DE USO .....	38
4.2.2 DIAGRAMA DE CLASSES .....	40
4.2.3 DIAGRAMAS DE SEQÜÊNCIA .....	42
4.3 DEMAIS MUDANÇAS NO TUTOR DELPHI .....	45
4.4 O TUTOR DELPHI NUMA VISÃO DISTRIBUÍDA.....	46
4.4.1 A CAMADA DE DADOS .....	49
4.4.2 A CAMADA DE REGRAS DE NEGÓCIOS .....	50
4.4.3 A CAMADA DE APLICAÇÃO.....	56
4.4.4 EXECUÇÃO DE UM CASO DE USO .....	59
5 CONSIDERAÇÕES FINAIS .....	63
5.1 CONCLUSÕES.....	63
5.2 DIFICULDADES ENCONTRADAS .....	64
5.3 SUGESTÕES .....	65
ANEXO I.....	66
GLOSSÁRIO.....	74
REFERÊNCIAS BIBLIOGRÁFICAS .....	76

## LISTA DE FIGURAS

Figura 1 - Arquitetura de UDA .....	11
Figura 2 - Integração dos componentes do OLE DB.....	15
Figura 3 - UDA Multi Plataforma .....	17
Figura 4 - Modelo de arquitetura de três camadas .....	20
Figura 5 - Arquitetura de aplicação MTS.....	21
Figura 6 - O Modelo Básico do ADO .....	24
Figura 7 - As tecnologias de acesso a dados alternativas disponíveis no Delphi 5.0.....	34
Figura 8 - Relação entre os agentes .....	37
Figura 9 – Diagrama de Casos de Uso .....	38
Figura 10 - Diagrama de Classes.....	40
Figura 11 - Diagrama de seqüência Cadastrar Usuário.....	42
Figura 12 - Diagrama de seqüência Cadastrar Exercício .....	43
Figura 13 - Diagrama de seqüência Aprender Exercício.....	44
Figura 14 - Diagrama de seqüência Corrigir Exercício.....	44
Figura 15 - Permissões do usuário.....	45
Figura 16 - Camadas do Tutor Delphi.....	47
Figura 17 - Modelo de Dados .....	49
Figura 18 - O Editor Type Library .....	50
Figura 19 - Módulo de Dados Remoto do ServidorTCC.....	54
Figura 20 - Configuração da Conexão ADO.....	55
Figura 21 - Módulo de Dados Cliente .....	57
Figura 22 - Propriedades do DCOMConnection .....	57
Figura 23 - Tela de acesso do usuário .....	59

Figura 24- Tela inicial do Tutor Delphi.....	60
Figura 25 - Entrada de novo usuário .....	61
Figura 26 - Log de acesso ao ServidorTcc .....	62



## LISTA DE TABELAS

Tabela 1 - Modelo de Componentes OLE DB .....	14
Tabela 2 - Características das camadas .....	20
Tabela 3 - Comparação entre sistemas antigos e atuais.....	29
Tabela 4 - Novos Recursos do Delphi 5.0.....	33
Tabela 5 - Mudanças básicas no Tutor Delphi .....	46

## LISTA DE QUADROS

Quadro 1 - Definição dos Métodos da Type Library.....	51
Quadro 2 - Definição da interface IUnknown .....	52
Quadro 3 - Criação da interface Servidor Tutor Delphi .....	52
Quadro 4 - String de conexão ADO .....	56

## LISTA DE ABREVIATURAS

ADO	– <i>ActiveX Data Objects</i>
API	– <i>Application Program Interface</i>
ASP	– <i>Active Server Page</i>
CGI	– <i>Common Gateway Interface</i>
CICS	– <i>Customer Information Control System</i>
COM	– <i>Component Object Model</i>
CORBA	– <i>Common Object Request Broker Architecture</i>
DAO	– <i>Data Access Objects</i>
DBMS	– <i>Database Manager System</i>
DCOM	– <i>Distributed Component Object Model</i>
DNA	– <i>Distributed Internet Application Architecture</i>
FTP	– <i>File Transfer Protocol</i>
HTML	– <i>Hypertext Markup Language</i>
HTTP	– <i>Hypertext Transport Protocol</i>
ISAM	– <i>Indexed Sequential Access Method</i>
ITE	– <i>Integrated Translation Environment</i>
MDAC	– <i>Microsoft® Data Access Component</i>
MIDAS	– <i>Middle-tier Application Services</i>
MSMQ	– <i>Microsoft® Message Queue Server</i>
MTS	– <i>Microsoft® Transaction Server</i>
ODBC	– <i>Open Database Engine</i>
OLE DB	– <i>Object Linking and Embedding - data base</i>
OODBMS	– <i>Object Oriented Database Manager System</i>

PC	– <i>Personal Computer</i>
RAD	– <i>Rapid Application Development</i>
RDBMS	– <i>Relational Database Manager System</i>
RDO	– <i>Remote Data Object</i>
RDS	– <i>Remote Data Service</i>
SQL	– <i>Structured Query Language</i>
UDA	– <i>Universal Data Access</i>
UML	– <i>Unified Modeling Language</i>

## RESUMO

Este trabalho tem como propósito demonstrar a utilização da tecnologia de manipulação de dados chamada *ActiveX Data Objects* (ADO), suas características e benefícios de sua utilização na implementação de sistemas, principalmente na manipulação de dados em sistemas com objetos distribuídos e pela Web. Visando demonstrar a utilização do ADO, será adaptado o acesso aos dados no Tutorial Inteligente para Delphi ([FRA1999]), utilizando esta tecnologia. Para isto foi utilizada a ferramenta visual Delphi 5.0

## **ABSTRACT**

This work aims to present the use of the data handle technology called *ActiveX* Data Objects (ADO), its features and advantages in the system development, mainly, the use of this technology in the distributed system development and Internet. In order to present the concepts and use of ADO technology, it is used to access data in a Delphi Intelligent Tutorial [FRA1999].

# 1 INTRODUÇÃO

Organizações de todos os tamanhos estão, atualmente, tendo a necessidade de criar soluções onde todos os dados possam ser considerados, pois até bem pouco tempo atrás, muitas informações eram desprezadas pelos sistemas, devido ao fato da dificuldade de integração de dados provindos de fontes diferentes, mas que em muitas vezes são de relevante importância nas decisões da organização, tanto a nível operacional quanto gerencial.

Dados e informação são o coração da maioria dos sistemas de computador, e o uso eficiente e efetivo da informação é o que concede valor e vantagem nas estratégias de negócios. Atualmente, este assunto é ainda mais amplo, pois as organizações de um modo geral estão redefinindo suas aplicações, implementando-as na internet e computação móvel, pois o acesso à informação requer novos cenários, e a complexidade de se obter essas informações e integrá-las cresce permanentemente. Organizações que mantinham seus dados em *mainframes*, agora podem obter conhecimento através de outras fontes, como o correio eletrônico, internet, sistemas distribuídos, bancos de dados diferentes, como Oracle, SQL Server ou outros. A tendência é de que as empresas que conseguirem migrar de suas arquiteturas de dados antigas e ultrapassadas, para sistemas com arquiteturas mais recentes e promissoras, como Cliente/Servidor, multicamadas e principalmente, a integração com a Internet, prosperarão.

A partir do conceito de *Universal Data Access*(UDA), que é uma proposta de uma interface única para acessar inúmeros tipos de dados, tanto relacionais quanto não-relacionais, pode-se conseguir esta integração das informações. Esta proposta partiu da *Microsoft*®, e utiliza tecnologias proprietárias desta empresa, ligando-a fortemente às plataformas Windows e ao “mundo *Microsoft*®”, o qual não se pode negar que é muito representativo no mercado.

UDA consiste em uma coleção de componentes de software que interagem e usam um conjunto comum de interfaces em nível de sistema, definida por *Object Linking and Embedding - Data Base* (OLE DB). OLE DB é uma programação de interface para fontes diversas de dados, que especifica um conjunto de componentes de comunicação que encapsulam vários serviços de administração de bancos de dados. Estes componentes permitem essa interface única. Considerando que OLE DB é uma interface de programação

em nível de sistema, o *ActiveX Data Objects* (ADO) é a interface em nível de aplicação ([MIC2000]).

Da mesma forma que o ODBC é programável através de objetos pelo *Remote Data Objects* (RDO), o OLE DB é programável através dos objetos do ADO. Esses objetos fornecem uma interface *Component Object Model* (COM) para acesso às funções das bibliotecas subjacentes, facilitando o uso por parte dos desenvolvedores de qualquer linguagem. O ADO uniformiza a interface de acesso a dados, tornando possível acessar, da mesma maneira, estruturas extremamente diferentes, como bases de dados SQL, mensagens de uma caixa postal Exchange, bancos de imagens e arquivos texto. Esse acesso é fornecido de uma maneira simples, orientada a componentes e compatível com o padrão de componentes COM ([CAL1999]).

ADO tem como propósito prover uma interface fácil de usar e pode ser implementado em Visual Basic, Java, C++ e Delphi. Como ADO é baseado em componentes *ActiveX*, somente pode ser utilizado em aplicações Windows, mas nada impede que essas aplicações acessem servidores que operem em outras plataformas, como Unix ou Linux, por exemplo.

Para demonstrar a capacidade da tecnologia ADO, será adaptado um programa que já foi desenvolvido em [FRA1999]. Este programa, que é um Tutorial Inteligente para Delphi, será aperfeiçoado, de forma a funcionar com objetos distribuídos. Os benefícios de adaptar o programa para este novo paradigma são a possibilidade de se utilizar vários bancos de dados nesta aplicação e a facilidade para o uso do programa, seja pela Web ou por uma rede local (como a dos laboratórios da FURB, por exemplo), sem a necessidade de cada aluno ter a base de dados do tutorial na sua própria máquina.

O resultado deste trabalho visa proporcionar uma maneira mais atrativa para o aprendizado da linguagem, além de demonstrar a utilidade do ADO, que é uma tecnologia emergente e com tendência de obter sucesso no mercado.

A especificação será feita sobre os conceitos já existentes, com algumas adaptações em nível de estrutura física das tabelas, utilizando uma metodologia orientada a objetos, representada através da UML - *Unified Modeling Language*. A ferramenta utilizada para esta especificação foi o Rational Rose, devido aos recursos disponíveis para aplicar as



representações da UML, como o Diagrama de Classes, o Diagrama de Casos de Uso e o Diagrama de Sequência.

Foi utilizado o ambiente de programação Borland Delphi 5.0, para a implementação do protótipo, pois este ambiente já suporta a tecnologia ADO.

## 1.1 OBJETIVOS

O objetivo principal do trabalho é implementar o uso da tecnologia ADO no protótipo do Tutorial Inteligente para Delphi.

Os objetivos secundários do trabalho são:

- a) demonstrar a tecnologia ADO;
- b) possibilitar o funcionamento do Tutorial Inteligente para Delphi, com objetos distribuídos ou pela Web.

## 1.2 ESTRUTURA

O trabalho foi estruturado da seguinte maneira:

O primeiro capítulo apresenta a contextualização e justificativa para o desenvolvimento da proposta do trabalho.

O segundo capítulo aborda a tecnologia ADO, suas características, alguns conceitos básicos sobre ADO e tecnologias auxiliares para utilização desta, como Objetos Distribuídos, *ActiveX*, *Universal Data Access* (UDA), OLE DB e a arquitetura Multicamadas. Todas estas tecnologias e conceitos, estão muito ligadas ao ADO, pois a implementação deste é baseada nestes paradigmas.

O terceiro capítulo será sobre a ferramenta de desenvolvimento utilizada para desenvolver o protótipo, o Borland Delphi 5.0, suas características e inovações, principalmente no que tange à implementação utilizando componentes ADO e objetos distribuídos.

O quarto capítulo descreve a especificação do protótipo, bem como detalhes de sua implementação.

O quinto capítulo apresenta as considerações finais, abrangendo as conclusões do desenvolvimento deste trabalho, as dificuldades encontradas e as sugestões para próximos trabalhos.

## 2 ACTIVE X DATA OBJECTS (ADO)

### 2.1 HISTÓRICO

Sempre houve uma certa dificuldade para possibilitar aos desenvolvedores de software uma independência do banco de dados, pois qualquer base de dados tem um modelo para armazenar seus dados, com tipos de dados específicos. As principais linguagens de programação disponibilizam componentes para fazer acesso aos dados, porém o modelo que o banco de dados utiliza para armazenar os dados e o modelo que alguns componentes de acesso a dados utilizam para armazená-los não precisam ser necessariamente os mesmos.

Componentes podem interoperar entre linguagens, ferramentas, sistemas operacionais e redes, mas apresentam características comuns aos objetos como polimorfismo e encapsulamento. Alguns componentes, chamados comumente de caixas pretas, não podem ser estendidos via herança, tais como componentes OLE. Componentes CORBA, no entanto, permitem herança.

Segundo [SES1998], um componente possui, no mínimo, as características a seguir :

- a) é um produto de mercado: deve ser empacotado de maneira que possa ser comercializável;
- b) não é uma aplicação completa: um componente deve ser combinado com outros para gerar uma aplicação. Deve ser implementado para realizar um número limitado de tarefas;
- c) pode ser usado em combinações imprevisíveis: deve poder ser usado de maneiras imprevistas pelo desenvolvedor;
- d) tem uma interface bem definida: assim como com objetos clássicos, os componentes só podem ser manipulados através de uma interface, que expõe suas funções ao mundo exterior;
- e) é um objeto interoperável: um componente pode ser invocado como um objeto através da rede, de linguagens de programação e sistemas operacionais. É uma entidade de software independente de sistemas;
- f) é um objeto expansível: os componentes suportam encapsulamento, herança e polimorfismo.

Em suma, componentes são partes de código reutilizáveis e independentes de aplicações e sistemas.

Em um modelo baseado em componentes, os dados são mantidos em uma camada que é responsável por apresentá-los num formato simples e eficiente para serem utilizados por componentes de acesso. Muitas camadas de componentes podem ser suportadas por uma determinada base de dados, e uma determinada camada pode suportar vários produtos de banco de dados diferentes. Isto permite às aplicações que utilizam estes componentes, algum grau de independência do banco de dados subjacente. Um exemplo de camada que permite essa independência é o ADO, pois baseia-se em componentes de acesso que possibilitam a transparência dos dados para a camada de interface da aplicação, permitindo que esta utilize qualquer fonte de dados. ([SES1998]).

ADO é uma tecnologia muito recente, que em cerca de 2 anos de existência, já tem uma estabilidade e tornou-se uma ferramenta de acesso a dados que pode ser usada em todos os tipos de desenvolvimento de aplicações.

O ADO é parte do *Microsoft® Data Access Components* (MDAC), o qual consiste no *ActiveX Data Objects* (ADO), no *Remote Data Service* (RDS), no *Microsoft® OLE DB Provider* para ODBC, no próprio ODBC e nos *drives* para diferentes base de dados. Pelo motivo de utilizar todas estas tecnologias originárias e proprietárias da *Microsoft®*, ADO somente pode ser utilizado em ambientes Windows.

Esta tecnologia pode ser implementada em Visual Basic, Java, C++ e Delphi. Na versão 5.0 do Delphi, estão disponíveis uma série de componentes que provêm recursos para o ADO.

Neste contexto, existe também uma estratégia para fornecer acesso a qualquer tipo de dados, chamada *Universal Data Access* (UDA - acesso a dados universal). A idéia é ter uma interface única que permita aos programadores usar suas ferramentas prediletas para acessar bancos de dados relacionais e outras fontes de dados menos estruturadas. O ADO é esta interface e por isso promete representar uma nova geração de acesso aos dados ([CAN2000]).

## 2.2 DEFINIÇÕES

Para entender melhor o que é ADO e como funciona, é necessário inicialmente conhecer alguns termos, conceituados em [MIC2000]:

- a) API (*Application Program Interface*): uma API é o método específico prescrito por um sistema operacional de computador ou por outro programa de aplicação, pelo qual um programador que escreve um programa de aplicação pode fazer pedidos ao sistema operacional ou outra aplicação;
- b) ASP (*Active Server Page*): ASP também é uma abreviação para provedor de serviço de aplicação. Uma página ASP é uma página HTML que inclui um ou mais *scripts* (pequenos programas embutidos) que são processados no servidor Web antes da página ser enviada ao usuário. Um ASP é parecido com um CGI, onde os programas rodam no servidor e normalmente constroem uma página para o usuário. Tipicamente, o *script* ASP recebe uma mensagem do requisitante, monta os resultados acessando a base de dados se necessário, customiza as informações e após envia-as ao requisitante. ASP é uma característica do *Microsoft® Internet Information Server (IIS)*, mas funciona em qualquer *browser*. Pode-se criar um arquivo de ASP incluindo um *script* em VBScript ou JScript em um arquivo de HTML ou usando *ActiveX Data Objects (ADO)*;
- c) DAO (*Data Access Objects*): foi a primeira interface orientada a objetos exposta para o *Jet database engine* (usado pelo *Microsoft® Access*) e permitiu aos desenvolvedores Visual Basic conectar tabelas do Access - e também outros bancos de dados - através do ODBC. DAO é melhor adaptado a aplicações de sistemas pequenos e monolíticos, onde a camada de regras de negócio e a camada de acesso a dados reside no mesmo executável;
- d) IIS (*Internet Information Server*): IIS é um conjunto de serviços, como Ftp, http, Web e outros serviços do sistema operacional WindowsNT. São programas para controlar e administrar *Web sites*, e suportam aplicações Web que têm interação com bases de dados. O uso do IIS resulta num servidor de páginas mais rápido e seguro;
- e) RDO (*Remote Data Objects*): é uma interface de acesso a dados orientada a objetos e que também fornece uma interface que expõe quase toda a flexibilidade e o poder do ODBC. O RDO é limitado, entretanto, pois ele não acessa muito bem bases de

dados Jet ou ISAM (como arquivos .DBF, por exemplo), e só pode ter acesso a bancos de dados relacionais através de drivers ODBC existentes. É mais utilizado com aplicações desenvolvidas em Visual Basic;

- f) RDS (*Remote Data Service*): é um serviço de transmissão de dados por uma rede. Por exemplo: se uma aplicação de rede proporciona para os clientes a habilidade para ter acesso a dados, pode-se distribuir o processo daqueles dados entre o cliente e o servidor com RDS. O componente cliente envia mensagens para o servidor da rede. O componente servidor RDS processa estes pedidos e os envia ao Sistema de Administração de Banco de Dados (DBMS) por uso de uma regra de negócios (camada intermediária da arquitetura utilizada pelo RDS - *Middle Tier*). O DBMS responde ao pedido e manda de volta os dados ao servidor de rede. Os componentes de RDS no servidor de rede transformam aqueles dados em um objeto ADO *Recordset* - conjunto de registros (mais adiante será explicado melhor cada um dos objetos ADO) . Os dados são analisados gramaticalmente para transporte ao cliente e são mandados de volta pela cadeia ao computador cliente, onde podem ser exibidos em controles como um texto ou *ComboBox*, por exemplo.

## 2.3 CONCEITO

*ActiveX Data Objects* (ADO) é um conjunto de componentes da *Microsoft*®, que provê, no nível de aplicação, uma interface de alto nível para habilitar os desenvolvedores a acessar dados e informação a partir de qualquer linguagem de programação em qualquer base de dados, seja local ou remotamente ([MAR1999]).

Essa interface, chamada ADO, é uma API *Microsoft*® que permite aos programadores, que escrevem aplicações com interface visual, obterem acesso para bancos de dados, relacionais ou não relacionais. Por exemplo, para escrever um programa que proporcionaria, para os usuários de uma rede local, dados de um banco IBM DB2 ou de um banco de dados Oracle, podem ser inclusos comandos ADO em um arquivo HTML que seria identificado, então, como um *Active Server Page* (ASP). Quando um usuário entrasse numa página, a página mandada de volta poderia incluir dados provindos de um banco de dados, utilizando código de ADO.

Como parte de *ActiveX*, ADO é também parte do modelo COM (*Component Object Model*), que é um modelo de programação baseado em objetos desenvolvido para promover interoperabilidade de software, ou seja, permitir que duas ou mais aplicações ou componentes cooperem facilmente entre si, mesmo que tenham sido desenvolvidos por diferentes fabricantes, em linguagens diferentes e até mesmo se estiverem rodando em sistemas operacionais diferentes([MIC2000]).

É ainda, uma evolução do RDO (*Remote Data Objects*). RDO trabalha com o ODBC da *Microsoft*® para ter acesso a bancos de dados relacionais, mas não bancos de dados não-relacionais. ADO também caracteriza *Remote Data Services* (RDS) pelo qual pode-se mover dados de um servidor para uma aplicação cliente, manipular os dados no cliente, e retornar atualização deste ao servidor em uma única viagem de ida-e-volta ([MIC2000]).

ADO provê consistência, acesso de alta performance para dados e suporte a uma grande variedade de necessidades de desenvolvimento, inclusive a criação de clientes de banco de dados *front-end* (2 camadas) e *middle-tier* (camada intermediária - modelo 3 camadas), usando aplicações, ferramentas, linguagens, ou *browsers* de Internet. ADO é projetado para ser uma interface de dados que tenha suporte para desenvolvimentos de sistemas implementados com multicamadas e implementações baseadas na Web.

ADO é implementado para funcionar com tráfego de rede mínimo e otimizar o transporte de dados em aplicações distribuídas. É projetado para combinar as melhores características RDO e DAO (*Data Access Objects*), e, eventualmente, até as substituir, usando convenções semelhantes, com semântica simplificada, para fazer de sua tecnologia um próximo passo natural para os desenvolvedores de software atuais.

Portanto, o ADO é projetado com o objetivo de ser a única interface de dados necessária para quaisquer tarefas de programação. Nem todos os programadores irão migrar para ADO, mas certamente esta é uma tecnologia a ser considerada ([CAN2000]).

Segundo [CAN2000] as principais vantagens do ADO são:

- a) a facilidade de uso;
- b) alta velocidade;
- c) baixa sobrecarga de memória;

- d) pouco gasto de disco;
- e) tráfego de rede mínimo nos principais cenários.

As características mais predominantes do ADO, de acordo com [MIC2000] são:

- a) a capacidade de suportar *stored procedures* com parâmetros de entrada, saída e devolução de resultados;
- b) permitir a otimização da comunicação;
- c) permitir o envio de mais do que um registo a partir de *stored procedures*.

ADO também pertence ao contexto do conceito de UDA (*Universal Data Access*), que é outro paradigma que está crescendo no meio da informática, devido às grandes vantagens que tende a proporcionar para o acesso aos dados, de uma maneira geral.

## 2.4 UDA - UNIVERSAL DATA ACCESS

UDA é uma proposta de uma interface única para acessar inúmeros tipos de dados, tanto relacionais quanto não-relacionais. É um novo preceito que, ao invés de promover a idéia de se tentar construir um banco de dados universal, provê acesso universal para vários tipos de bancos de dados. Para que isto se torne realidade, a *Microsoft*® e outras companhias de banco de dados construíram um "programa de ponte" entre o banco de dados e o OLE DB da *Microsoft*®. OLE DB é o serviço de sistema subjacente que um programador que usa ADO está usando de fato.

Hoje, companhias que constróem soluções Cliente/Servidor de banco de dados baseado em redes buscam máxima vantagem empresarial dos dados e informação distribuídas ao longo das suas organizações. UDA provê acesso de alto-desempenho para uma variedade de dados e fontes de informação em plataformas múltiplas e uma interface de programação fácil de usar que trabalha com praticamente qualquer ferramenta ou linguagem, alavancando as habilidades técnicas que o programador já têm. As tecnologias que suportam UDA permitem às organizações criar soluções fáceis de manter, com os dados nos clientes, na camada intermediária ou no servidor ([MIC2000]).

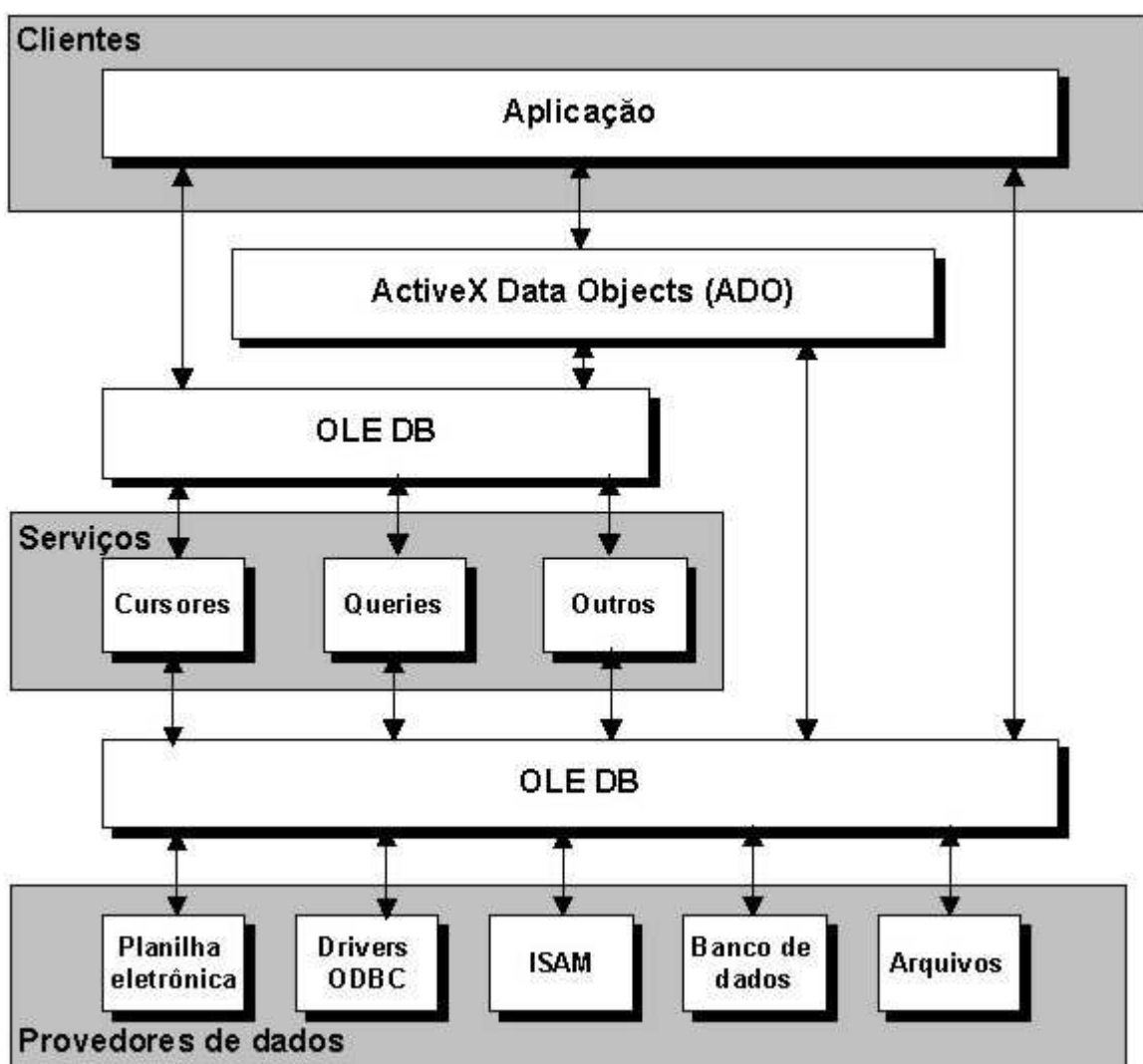
Outro benefício da UDA é que não requer movimento caro e demorado de todos os dados incorporados em um único banco de dados, nem requer compromisso com produtos de



um único vendedor. UDA está baseada em especificações abertas e trabalha com os principais produtos de banco de dados estabelecidos. UDA é um passo evolutivo de interfaces de padrão como ODBC, RDO, ADO, e estende significativamente a funcionalidade destas tecnologias ([MIC2000]).

A figura 1 ilustra a arquitetura de UDA.

**Figura 1 - Arquitetura de UDA**



Fonte: ([MIC2000])

Provedores de dados são componentes que representam fontes diversas de dados como bancos de dados, arquivos indexados, planilha eletrônicas e arquivos de correio eletrônico. Os provedores expõem uniformemente os dados utilizando uma abstração comum chamada *rowset* (conjunto de linhas, registros).

Serviços, ou regras, são componentes que estendem a funcionalidade dos provedores de dados implementando interfaces que não são nativas do banco de dados.

Clientes são componentes que utilizam OLE DB. Exemplos de clientes incluem modelos de acesso a dados de alto nível, como ADO, aplicações empresariais escritas em Visual Basic, C++, Delphi ou Java e ferramentas de desenvolvimento.

Todas as interações entre componentes da figura 1, que são indicadas através de setas bidirecionais, podem ultrapassar o limite de máquinas por protocolos de rede, como *Microsoft® Distributed COM* (DCOM) ou HTTP. Interações entre componentes de transações são possíveis por um serviço de coordenação de transações distribuídas como o MTS (*Microsoft® Transaction Server*). No item 2.7 deste trabalho, onde será descrita a arquitetura de três camadas, poder-se-á verificar um exemplo da arquitetura de uma aplicação que utiliza MTS em suas transações.

Segundo um artigo de David Lazar, publicado em [MIC2000], uma força da estratégia UDA é que é integrada por um conjunto de modernas interfaces orientadas a objeto. Estas interfaces, bem como MTS, estão baseado no modelo COM, que provê o seguinte:

- a) os serviços integrados de transações, segurança e acesso a dados para suportar os diversos cenários das aplicações;
- b) a escolha de qualquer linguagem de programação;
- c) suporte para aplicações de customização e componentes reutilizáveis;
- d) interoperabilidade.

Por causa da consistência e interoperabilidade previstas pela COM, a UDA tem sua arquitetura aberta e trabalha com virtualmente qualquer ferramenta ou linguagem. Também habilita a consistência dos dados em todas as camadas do modelo de arquitetura multicamadas. No nível de aplicação, os componentes ADO é que fazem a interface.

### 2.4.1 ADO E UDA

UDA realmente é um passo evolutivo para o padrão de dados atualmente discutido. É conhecida a “sopa de letrinhas” - ODBC, RDO, MTS e DAO. UDA é um passo para estender a funcionalidade destas tecnologias. O pacote de tecnologias que compõem UDA consiste em *ActiveX Data Objects* (ADO), *Remote Data Service* (RDS, antigamente conhecido como Conector de Banco de Dados Avançado ou ADC), OLE DB, e ODBC. Juntas, estas interfaces provêm os meios para trabalhar com qualquer fonte de dados. E juntas elas são conhecidos como *Universal Data Access*.

## 2.5 OLE DB

Por si só, ADO não é capaz de qualquer coisa. Para prover alguma funcionalidade, ADO precisa dos serviços de um provedor OLE DB.

OLE DB é a estratégia de programação que faz a interface com os dados. É uma especificação aberta projetada para interagir com ODBC provendo um padrão aberto para se ter acesso a todos os tipos de dados ([MIC2000]).

Numa outra terminologia, OLE DB pode ser visto como um conjunto de componentes compostos de fornecedores de dados, que contêm e expõem dados, consumidores de dados, que usam os dados, e componentes de serviço, que processam e transportam dados (como os processadores de consultas e mecanismos de cursor) ([CAN2000]).

OLE DB define uma coleção de interfaces COM que encapsulam vários serviços de administração de sistemas banco de dados. Interfaces OLE DB são projetadas para ajudar a integrar suavemente os componentes clientes para proporcionar alta qualidade e possibilidade rápida de comercialização. Além disso, OLE DB inclui uma ponte para ODBC que habilita suporte continuado para a grande quantidade de *drivers* ODBC disponível hoje.

O modelo de objetos OLE DB consiste de sete componentes principais, conforme a tabela 1:

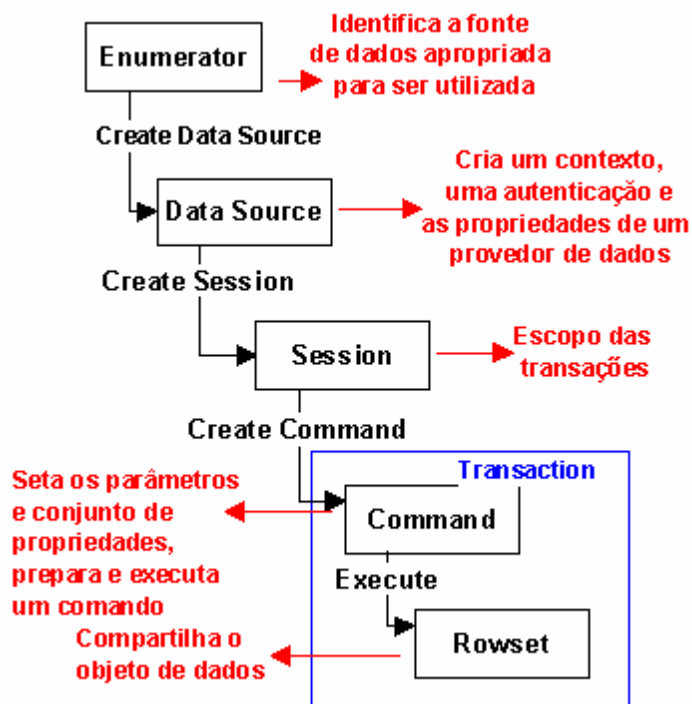
**Tabela 1 - Modelo de Componentes OLE DB**

<b>Componente</b>	<b>Descrição</b>
<i>Enumerator</i>	<i>Enumerators</i> é uma enumeração, um identificador para uma fontes de dados. Clientes que não estão customizados para uma base de dados específica utilizam um <i>enumerator</i> para procurar uma fonte de dados para usar.
<i>Data Source</i>	Objetos <i>Data Source</i> contêm as implementações para conectar a uma fonte de dados, como um arquivo ou um DBMS. Há uma fábrica de <i>sessions</i> neste objeto.
<i>Session</i>	<i>Sessions</i> provêem um contexto para transações( <i>transactions</i> ) e podem ser implícita ou explicitamente manejadas. Um único objeto <i>Data Source</i> pode criar sessões múltiplas. <i>Sessions</i> são uma fábrica para <i>transactions</i> , <i>commands</i> , e <i>rowsets</i> .
<i>Transaction</i> (transações)	Objetos <i>Transaction</i> são usados quando se está executando ou abortando transações aninhadas a outras de mais baixo nível.
<i>Command</i>	<i>Command</i> executam um texto de comando, como uma instrução SQL. Se o comando especifica um <i>rowset</i> , como uma instrução SELECT de SQL, o comando é uma fábrica para o <i>rowset</i> .
<i>Rowset</i>	<i>Rowsets</i> expõem dados em formato tabular.
<i>Error</i>	Objetos <i>Error</i> podem ser criados por qualquer interface em qualquer objeto OLE DB. Eles contêm informação adicional sobre um erro, incluindo, opcionalmente, um objeto de erro customizado.

Fonte: ([MIC2000])

Os componentes OLE DB interagem entre si, conforme a figura 2:

**Figura 2 - Integração dos componentes do OLE DB**



O objeto *Data Source* é o primeiro objeto instanciado por um *enumerator* e contém a estrutura necessária para conectar à fonte de dados subjacente.

O objeto *Session* controla o escopo de transações e é responsável pela geração dos dados (*rowsets*) da fonte de dados (*Data Source*) subjacente. Podem ser associadas múltiplas sessões (*Sessions*) com um único objeto *Data Source*. Se o provedor OLE DB suporta comandos, o objeto *Session* também age como uma fábrica de classe para objetos *Command* (fábricas de classe são responsáveis pela geração de objetos COM sob requisição).

Se o provedor OLE DB suporta *queries*, também tem que expor um objeto *Command*. Pode haver vários objetos *Command* associados com um único objeto *Session*. O objeto *Command*, gerado pelo objeto de *Session*, é responsável por especificar, preparar e executar comandos SQL. O objeto pode ser criado por qualquer um dos outros objetos, caso ocorra algum erro.

ADO utiliza um conjunto de objetos (ver o modelo básico de ADO no item 2.8) para prover uma interface muito simples e eficiente com OLE DB que, em troca, provê acesso para uma variedade muito grande de tipos de dados. Isto faz de ADO uma linguagem virtualmente independente.

### 2.5.1 ADO E OLE DB

Muitos podem estar se perguntando porque a *Microsoft*® introduziu a camada ADO, em vez de permitir que os programadores acessem o OLE DB. O principal motivo é a complexidade do OLE DB. O ADO encapsula mais de 60 interfaces OLE DB em cerca de 20 objetos simples. Quando se utiliza ADO, não é preciso preocupar-se com interfaces, alocações de memória, contagem de referências ou fábrica de classes (todos os quais são problemas do OLE DB) ([CAN2000]).

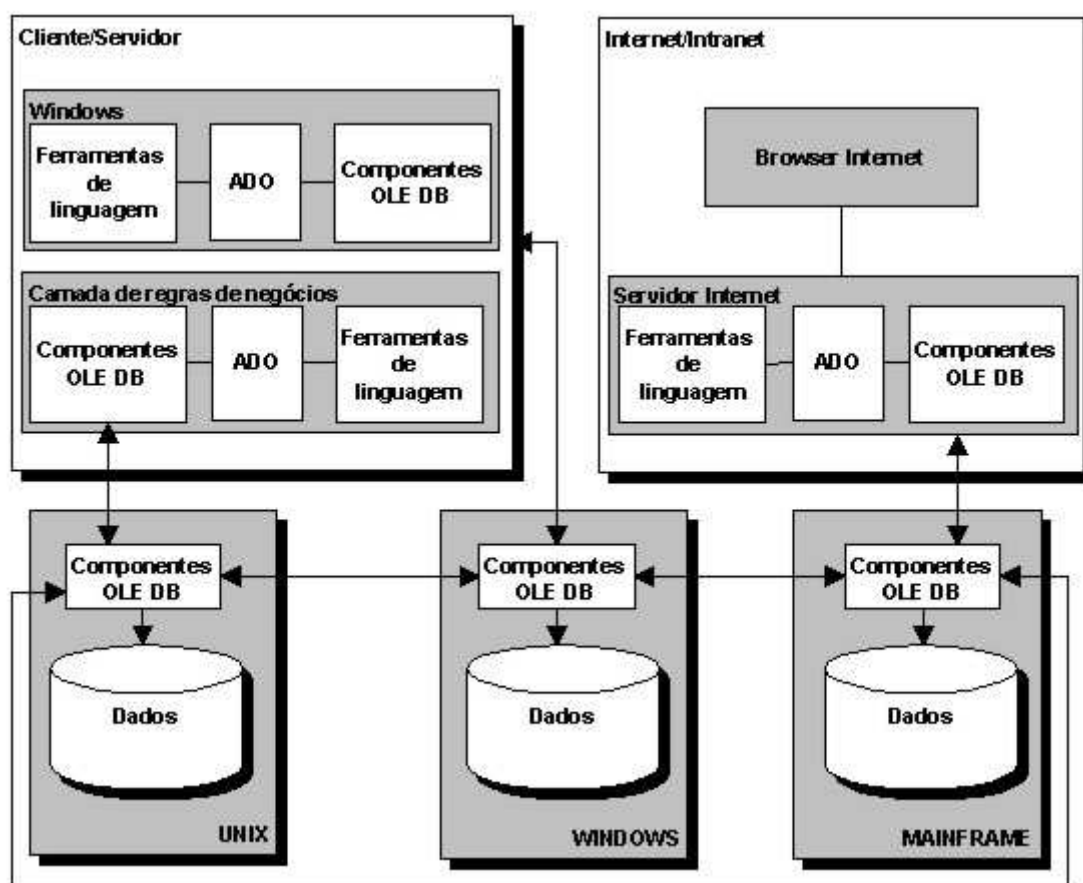
Uma resposta alternativa é que o ADO tem por objetivo as ferramentas de programação de alto nível, como Visual Basic, que não conseguem tratar com interfaces COM através de objetos de nível mais baixo. Em vez disso, foram criados alguns objetos *ActiveX* que empacotam a funcionalidade OLE DB, expondo aos programadores apenas os conceitos mais importantes de uma maneira mais simples e fácil de usar. Com ADO, todas as ferramentas que oferecem suporte a tecnologia *ActiveX* (sejam elas compiladores completos ou linguagens de *scripts* simples) podem se associar ao modelo UDA.

OLE DB é, então, um conjunto de interfaces de baixo nível que provêem acesso para dados em uma variedade de formatos e localizações. ADO provê uma interface amigável para implementadores para fazer com que o trabalho com OLE DB e UDA seja uma tarefa relativamente fácil. Não importa qual o ambiente de programação que se utilize, a interface usada para se ter acesso aos dados permanece constante. Esta interface é ADO que em contrapartida usa OLE DB.

## 2.6 ADO E OLE DB COMO INTEGRANTES DE UDA MULTI PLATAFORMA

Enquanto o sistema operacional WindowsNT (e também já o seu sucessor Windows2000) estão emergindo como plataformas importantes para administração de banco de dados, muitas organizações confiam em uma mistura de sistemas operacionais e plataformas de banco de dados. Para obter êxito, qualquer estratégia de acesso a dados deve ter performance igual e eficiência em todas as principais plataformas. UDA provê o fundamento para suportar o acesso eficiente a dados nas principais plataformas de computação atuais. Produtos que usam a arquitetura UDA para ter acesso aos principais gerenciadores de banco de dados, atualmente já conseguem fazê-lo em plataformas diferentes do Windows. Na figura 3 pode ser vista como seria esta interação.

**Figura 3 - UDA Multi Plataforma**



Fonte: ([MIC2000])

Isto é possível porque a especificação de OLE DB define componentes de interfaces que possibilitam a portabilidade em outros sistemas operacionais. OLE DB está baseado na arquitetura COM que é um modelo de objetos Windows. Isto poderia implicar que componentes OLE DB rodariam somente em sistemas Windows, porém, surpreendentemente não é isso. OLE DB tem dois enfoques separados que provêm portabilidade para plataformas de sistemas gerenciadores de banco de dados diferentes do Windows ([MAR1999]).

A ampla disponibilidade que a arquitetura UDA possibilita para as organizações que suportam plataformas de sistemas de gerenciador de banco de dados múltiplas, é mais um fato que evidencia que os sistemas que utilizarem OLE DB e ADO terão um benefício adicional. Mas é importante salientar que em nenhum momento foi referenciado uma camada de aplicação em outras plataformas diferentes do Windows. Todos os documentos analisados até aqui, sempre mencionam a multiplicidade de plataforma na camada de dados, e não da de aplicação.

## 2.7 ARQUITETURAS DE CAMADAS

Quando se desenvolve uma aplicação utilizando ADO, não é preciso obrigatoriamente modelar esta aplicação em três camadas.

Apesar da popularidade das aplicações em três camadas estar crescendo, ainda há muito espaço para aplicações em duas camadas. Então, quais os critérios para a escolha de uma ou outra opção? Segundo [MIC2000], deve-se adotar a arquitetura em três camadas para aplicações com as seguintes características:

- a) mais de 50 serviços ou classes;
- b) aplicações programadas em diferentes linguagens;
- c) duas ou mais fontes de dados heterogêneas, tais como dois gerenciadores de banco de dados;
- d) expectativa de vida de mais de três anos para a aplicação, especialmente quando são previstos novas versões do sistema;
- e) alto volume de transações (mais de 50.000 por dia ou mais de 300 usuários acessando o sistema concorrentemente);
- f) necessidade de comunicação com diferentes aplicações.



Apesar dessas características levarem a crer que somente grandes e complexas aplicações são candidatas a serem desenvolvidas usando os conceitos de três camadas, nada impede o uso desses mesmos conceitos em aplicações menores, principalmente quando estas aplicações envolvem objetos distribuídos.

Seguindo a história dos modelos de camadas das aplicações, verifica-se que inicialmente, os aplicativos de bancos de dados para PC eram soluções apenas de clientes: o programa e os arquivos de dados ficavam no mesmo computador. Num segundo momento, alguns programadores já disponibilizaram um único arquivo de dados, na rede. Os computadores clientes ainda continham o software aplicativo e o mecanismo de banco de dados inteiro, mas os arquivos de bancos de dados eram, então, acessíveis a vários usuários simultaneamente ([CAN2000]).

A próxima grande transição foi o desenvolvimento de cliente/servidor. Este modelo, que é baseado em duas camadas, tem a maioria da sua lógica rodando no cliente, que envia requisições SQL para o gerenciador de banco de dados que reside no servidor.

Na arquitetura em duas camadas, a lógica da aplicação pode estar localizada tanto na interface, com um cliente, quanto no gerenciador de banco de dados no servidor. A interface roda no cliente, que envia, via SQL, chamadas a arquivos de sistema ou comandos HTTP através da rede para o servidor, que então processa a requisição e retorna os resultados. Para acessar os dados, os clientes precisam estar cientes de como os mesmos estão organizados e armazenados no servidor. Uma variação da abordagem em duas camadas usa procedimentos implementados no próprio gerenciador de banco de dados (*stored procedures*) para processar requisições SQL no servidor, reduzindo o tráfego de rede.

A grande popularização da arquitetura em duas camadas deve-se sobretudo à disponibilidade cada vez maior de ferramentas de desenvolvimento rápido (RAD) no mercado e na sua simplicidade. O Visual Basic e o Delphi são os atuais líderes nesse segmento de mercado ([CAN2000]).

Apesar de, à primeira vista, poder se achar que colocar todo o código ou boa parte dele no cliente seja uma boa solução devido ao poder cada vez maior de processamento dos microcomputadores, uma análise mais detalhada na maneira como as aplicações empresariais

e as regras de negócio realmente se comportam, pode levar a uma conclusão totalmente diferente.

As corporações estão migrando para uma arquitetura cliente/servidor em três camadas para compensar as limitações da arquitetura em duas camadas. A camada adicional fornece um espaço para as regras de negócio. Essa camada intermediária encapsula o modelo do negócio associado ao sistema e o separa da interface com o cliente e do banco de dados.

A arquitetura em três camadas está representada na figura 4:

**Figura 4 - Modelo de arquitetura de três camadas**



Fonte: [MIC2000]

Na tabela 2, pode-se verificar as características de cada camada:

**Tabela 2 - Características das camadas**

Camada	Tipo de Serviço	Característica	Responsável Por
Interface com o usuário (Apresentação)	Aplicações cliente	Interface	Apresentação e navegação
Regras de negócios	Servidores de regras de negócio	Objetos de negócio	Políticas de negócio, regras e segurança
Banco de dados	Servidores de bancos de dados	Gerenciadores de dados	Integridade dos dados

A primeira camada (serviços do usuário) fornece a interface visual para apresentação das informações e aquisição de dados. Essa camada contém vários componentes aptos a representar e manipular dados.

Normalmente, o desenvolvimento da camada de serviços de usuários é dividido em duas partes:

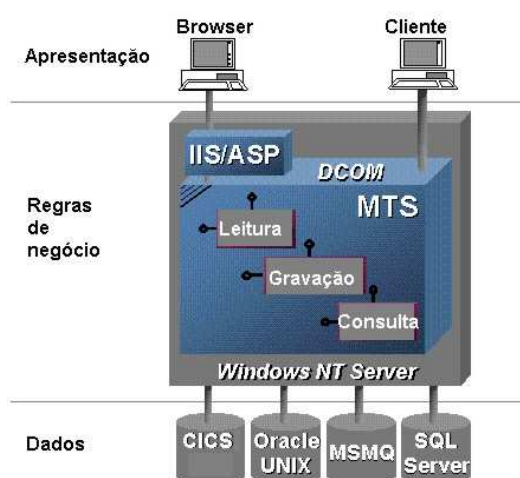
- a) desenvolvimento do *layout* da interface, gráficos e mensagens mostradas ao usuário. Isso é geralmente atingido com a construção do protótipo da aplicação sem o uso de objetos de negócios;
- b) criação do código específico da aplicação no cliente. O desenvolvedor usa normalmente uma linguagem de alto nível e os pontos de entrada para os vários serviços de negócios para criar o fluxo de controle requerido pela aplicação. O desenvolvedor preocupa-se mais com a aplicação do que com a complexidade da modelagem do negócio.

A camada de serviços de negócio, segunda camada, é uma ponte entre o usuário e o serviço de banco de dados. Ela responde a requisições dos usuários para a execução de uma tarefa, através de procedimentos e regras de negócio.

A terceira camada (serviços de banco de dados) contém, conceitualmente, uma coleção de dados usados na camada de serviços de regras de negócio para tomada de decisões. Geralmente é composta por um sistema gerenciador de banco de dados ou por uma coleção heterogênea de bancos de dados que residem em diferentes plataformas.

Na figura 5 pode-se verificar como é a arquitetura de uma aplicação que utiliza MTS na camada intermediária, no nível de transações.

**Figura 5 - Arquitetura de aplicação MTS**



Fonte: ([MAR1999])

No exemplo da figura 5, é demonstrado uma aplicação com três camadas, baseada no modelo COM. Em aplicações MTS, componentes de lógica de aplicação são executados sob controles MTS que são instanciados por componentes da camada intermediária, os quais interligam-se aos clientes pelas tecnologias COM ou DCOM, quando em ambientes distribuídos. Clientes podem ser aplicações convencionais, ou página ASP que funcionam sob servidores IIS.

Os componentes de regras de negócio (ou lógica de aplicação para alguns autores) podem ter acesso a várias fontes de dados diferentes, como MSMQ (*Microsoft® Message Queue Server*), CICS (*Customer Information Control System*,) e outros.

Tradicionalmente, a comunicação somente é feita entre camadas adjacentes ou internamente. No entanto, aplicações em três camadas não necessariamente requerem hardware em três camadas. Por exemplo, alguns dos componentes da camada intermediária podem ser implementados no mesmo servidor do banco de dados.

A camada intermediária da maioria das aplicações em três camadas são compostas de uma coleção de componentes que são usadas em uma variedade de transações de regras de negócio invocadas pelos clientes. Cada componente automatiza uma função do negócio.

Objetos distribuídos são, por definição, componentes. Em sistemas de objetos distribuídos, a unidade de trabalho e distribuição é o componente.

Os serviços são compostos de componentes, que agem no contexto de uma camada. Enquanto um serviço é um conceito lógico, um componente descreve, através de código, uma funcionalidade do negócio. Clientes geralmente utilizam-se de diversos componentes em uma única transação.

Quando se desenvolve a camada intermediária utilizando-se componentes, obtém-se os seguintes benefícios :

- a) a arquitetura baseada em componentes permite desenvolver aplicações críticas decompondo-as em aplicações menores, facilitando o desenvolvimento;
- b) aplicações podem reutilizar componentes como se fossem caixas pretas;
- c) os clientes não necessitam saber qual o banco de dados que está sendo acessado, nem qual o componente que está atendendo sua solicitação;

- d) pode-se incorporar componentes externos para ir “montando” uma aplicação. Obviamente isso só é possível através da adoção de algum padrão comum de comunicação entre os componentes.

Mas onde entra o ADO nisso tudo?

ADO é justamente a tecnologia que visa fazer a ponte entre a camada da aplicação e a camadas de regras de negócios, se existir esta camada. Caso contrário, ADO será a ponte direta entre a aplicação e a camada de dados. Os componentes ADO fazem essa ponte ([MIC2000]).

Mas é preciso atentar para a seguinte questão: quando se utiliza ADO em programas modelados em três camadas e há alguma requisição que precise ser feita para a camada de dados, ADO faz o elo entre a camada de aplicação e a fonte de dados e, se necessário for, irá passar pela camada intermediária antes de chegar à camada de dados, fazendo uso dos métodos ali implementados, sempre interagindo também com OLE DB ([MIC2000]).

## 2.8 MODELO BÁSICO DE PROGRAMAÇÃO ADO

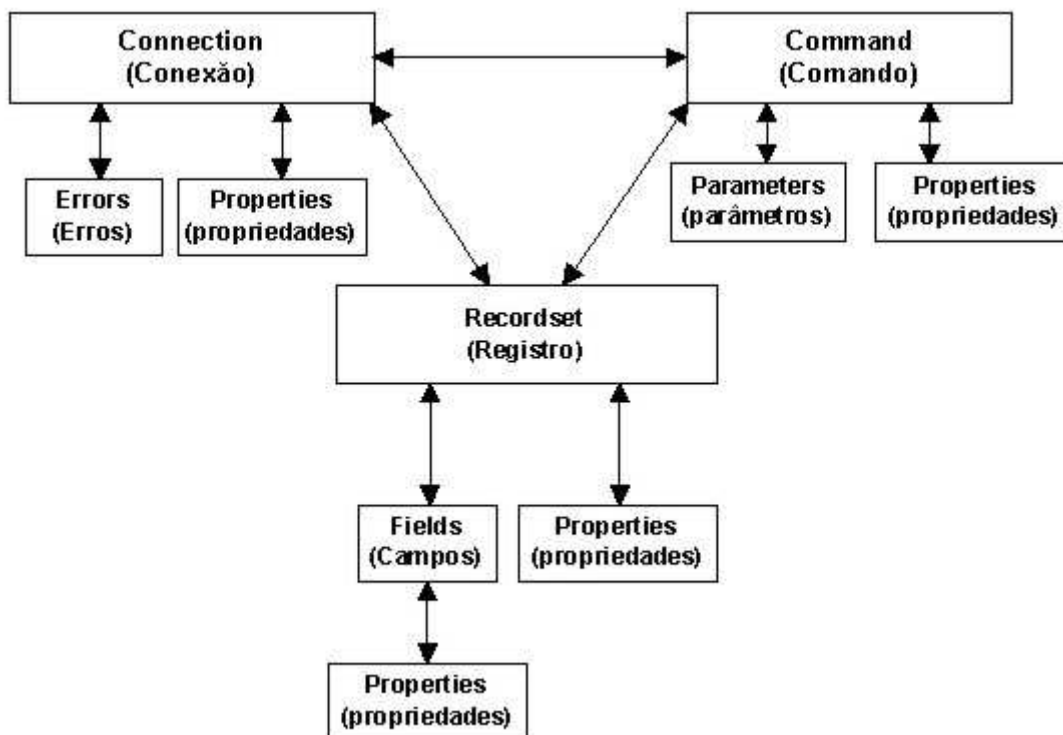
Basicamente, para utilizar-se ADO a seguinte sucessão de ações é suficiente:

- a) conectar a uma base de dados. Pode-se assegurar que a conexão obteve sucesso;
- b) especificar um comando (ação), com parâmetros ou não;
- c) executar o comando;
- d) se o comando retorna dados, estes são armazenados numa estrutura particular, onde se possa facilmente alterar, inserir ou excluir informações;
- e) se for apropriado, atualizar os dados armazenados (*refresh*);
- f) prover uma maneira de tratamento de exceções, principalmente após tentativa de conexão ou de executar algum comando

Tipicamente, todo esse roteiro é utilizado, porém algumas vezes parte dele é suficiente. Por exemplo: pode-se armazenar as informações após a conexão e utilizá-las somente para consultas.

Na figura 6 é apresentado graficamente o modelo básico de ADO:

**Figura 6 - O Modelo Básico do ADO**



Fonte: [MIC2000]

Os objetos *Connection*, *Command* e *Recordset* podem ser criados e destruídos independentemente de cada um dos outros objetos. Eles não tem herança entre si. Pode-se dizer que são objetos irmãos. Embora possa ser criado o objeto *Parameter* independentemente de um objeto *Command*, ele deve ser associado a um objeto *Command* antes de ser utilizado. *Field*, *Error* e *Property* são objetos que existem somente dentro do contexto do objeto que os implementa(objeto pai), e não pode ser criado separadamente.

O objeto *ADO Connection* encapsula os objetos fonte de dados do OLE DB (*Data Source*). Ele representa uma única sessão com a fonte de dados. O objeto *Connection* define propriedades da conexão, nomeia o escopo de transações locais, provê uma localização central para tratar erros, e provê um ponto para executar *queries*.

O objeto *ADO Command* é equivalente ao objeto *OLE DB Command*. O *Command* especifica a definição de dados ou declaração de manipulação dos dados a ser executada. No caso de um provedor de dados relacionais, esta é uma instrução SQL. O objeto *Command* permite especificar parâmetros e customizar o comportamento da declaração a ser executada. Um conjunto de objetos *Parameter* expõe esses parâmetros. O objeto *Parameter* representa parâmetros de um comando.

O objeto *ADO Recordset* encapsula o objeto *OLE DB Rowset*. O objeto *Recordset* é a interface atual aos dados. Podem ser dados resultantes de uma *query* ou gerados de outra forma. O objeto *Recordset* provê controle em cima do mecanismo usado, do tipo de cursor usado, do número de linhas para ter acesso de cada vez, e assim por diante. O objeto *Recordset* expõe um conjunto de objetos *Field* que contêm os dados sobre as colunas no *Recordset*, como o nome, tipo, duração, e precisão, como também os próprios dados. *Recordset* é utilizado para navegar por registros de dados. A interface *Field* representa uma coluna em um *Recordset* que pode ser usado para obter e modificar valores. Esta interface é quase idêntica a modelos passados.

A interface *Error* representa um objeto de erro retornado de uma fonte de dados. Esta interface é opcional e só é útil quando o provedor de dados subjacente pode retornar erros múltiplos. Se o provedor de dados está impossibilitado de devolver erros múltiplos para uma única chamada de função, o provedor usará o mecanismo COM normal de erros, ou seja, as mensagens de erros normais baseadas na tecnologia COM.

Cada um dos três objetos principais ADO (*Command*, *Connection* e *Recordset*) contêm um conjunto de objetos *Properties*. O objeto *Properties* permite que ADO exponha dinamicamente as capacidades de um provedor específico. Pelo fato de nem todos os provedores de dados terem a mesma funcionalidade, é importante que ADO modele isso, para permitir o acesso de modo dinâmico em provedor de dados específicos. Isto também previne que o modelo ADO fique desordenado, cheio de propriedades que só estão disponíveis em certas circunstâncias ([MIC2000]).

## 2.9 APLICAÇÕES DISTRIBUÍDAS COM ADO

A tecnologia ADO, diante do conceito que lhe é atribuído, é perfeita para ser utilizada em aplicações distribuídas. Sua principal característica, que é a independência da fonte de dados, também é muito útil quando trata-se de aplicações distribuídas, pois permite que uma mesma interface (objeto que interage com o usuário), acesse a camada intermediária (e por consequência a de dados), seja esta última de que fonte for.

Resumidamente, se o usuário de um aplicação distribuída que utiliza ADO no acesso aos dados quiser mudar sua fonte de dados, pode fazê-lo sem alterar o objeto de interface. Basta fazer as devidas mudanças na conexão de seus dados com os componentes ADO. Os componentes ADO, fisicamente, são implementados no programa servidor dos objetos distribuídos, porém isso não quer dizer que eles deixam de ser uma interface a nível de aplicação, pois eles continuam fazendo a ponte entre a camada de aplicação e a camada intermediária, enviando as informações oriundas da fonte de dados, via componentes específicos para essa função, para a camada de aplicação. ADO é o elo entre a fonte de dados e a aplicação, mas, numa modelagem em três camadas, ADO fica no nível mais alto, entre a aplicação e a camada intermediária, e OLE DB fica num nível mais baixo, fazendo a ligação com a camada de dados.

Utilizando ferramentas de programação que já suportam a tecnologia ADO, como é o caso do Delphi 5.0, do Visual Basic e do C++, por exemplo, pode-se construir aplicações distribuídas utilizando os próprios recursos da ferramenta, e associando a programação distribuída com ADO.

Porém, antes de mais nada, é preciso conhecer um pouco mais de objetos distribuídos e de *ActiveX*.

### 2.9.1 OBJETOS DISTRIBUÍDOS

Hoje em dia o termo objeto é usado por muitas pessoas que quase nunca identificam sobre qual campo estão se referindo, o que causa grande confusão. Existem, pelo menos, quatro campos diferentes em que a orientação de objetos é importante:

- a) linguagens de programação orientada a objeto: nas linguagens de programação orientadas a objetos os programadores podem escrever um código usando um



modelo de objetos em vez dos modelos da programação mais tradicional, em que o código e os dados são entidades separadas;

- b) software de sistemas orientados a objetos: o software de sistemas orientados a objetos fornece os blocos básicos para manipular objetos no nível do sistema operacional. Sistemas de objetos permitem que objetos criados por empresas diferentes, talvez utilizando linguagens de programação diferentes, trabalhem juntos;
- c) bancos de dados orientados a objetos: são sistemas gerenciadores de banco de dados que substituem o modelo de banco de dados relacional por um modelo orientado a objetos. Um banco de dados orientados a objetos é projetado para armazenar dados e métodos de objetos, com técnicas que são eficientes para o processamento baseado em objetos;
- d) objetos distribuídos: o software de sistemas orientados a objetos permite que objetos, em sistemas diferentes em rede, trabalhem juntos, ou seja, os objetos podem se comunicar entre si, mesmo que não estejam no mesmo computador;

Segundo [HON1999], pode-se diferenciar os objetos clássicos dos objetos distribuídos através das seguintes características:

- a) um objeto clássico possui propriedades e métodos e é gerado através de uma linguagem de programação como C++, Smalltalk, Delphi, entre outras. Esses objetos fornecem reusabilidade de código através de herança, encapsulamento e polimorfismo, propriedades fundamentais da teoria orientada a objetos. Contudo, esses objetos só vivem dentro de um programa e apenas o compilador que os criou conhece a sua existência. O mundo externo os desconhece e não tem forma de acessá-los;
- b) um objeto distribuído também é uma classe que pode publicar (tornar disponível) tanto suas propriedades quanto os seus métodos, mas a linguagem e compilador usados para criar objetos distribuídos são totalmente transparentes para a implementação desses. Esses objetos têm uma interface definida, onde os compiladores geram um código a mais para serem acessados por outros objetos de forma totalmente transparente, isto é, o invocador desconhece o local do objeto invocado, o sistema operacional que esse executa, podendo estar fisicamente na mesma máquina (processo) ou em alguma máquina remota. Objetos distribuídos

podem ser implementados utilizando-se tecnologias como *Distributed Component Object Model* (DCOM) e *Common Object Request Broker Architecture* (CORBA), que são os mais difundidos.

Segundo ([HON1999]) as necessidades que existiam e que ainda existem, que levaram os pesquisadores e os empresários a investirem tempo e recursos no desenvolvimento dos objetos distribuídos e as vantagens e avanços que esse conceito oferece têm íntima relação com os fatores de reutilização e heterogeneidade.

A programação e o desenvolvimento de software têm sido uma questão de reinventar o que já existe. Toda vez que surge a necessidade de se construir um modelo, precisa-se refazer todos os passos já existentes. Com isso, o desafio para a maioria dos programadores tem sido justamente o parar de reinventar, construir um modelo padrão e a partir desse modelo padrão ter condições de implementar melhorias, desenvolvendo, desta forma, o conceito de reutilização.

No que se refere a heterogeneidade, existe cada vez mais a necessidade das aplicações serem desenvolvidas independentemente de linguagens, de plataformas de hardware, de software e precisarem compartilhar informações e invocar serviços umas das outras.

A princípio criaram-se as sub-rotinas que serviram como um artifício para construção desse padrão. O objetivo era a criação de uma biblioteca de sub-rotinas, as quais só precisariam ser codificadas e testadas uma única vez, mas que pudessem ser reutilizáveis em qualquer programa que solicitasse a mesma função.

Entretanto, as sub-rotinas só apresentaram um bom funcionamento em funções triviais, além disso, não auxiliaram muito na criação de software reutilizável.

Atualmente, existe uma nova etapa na evolução da informática com a utilização dos objetos, os quais surgem como a primeira oportunidade real de reutilização e heterogeneidade de software, e que, quando disponíveis em rede, dão origem aos objetos distribuídos.

A tabela 3 demonstra como é evidente a situação do mercado:

**Tabela 3 - Comparação entre sistemas antigos e atuais**

<b>Sistemas Antigos</b>	<b>Sistemas Atuais</b>
Ambiente Centralizado	Ambiente Cliente/Servidor
Forte concentração no processamento centralizado	Concentração na integração e distribuição do processamento entre os computadores da rede
Terminais burros, modo caracter	Terminais com poder de processamento e interfaces gráficas intuitivas e de fácil uso
Dificuldade para o crescimento por limites na capacidade de processamento e alto custo do computador central	Facilidade de crescimento por permitir que o ambiente seja escalável e a um custo linear
Falta de integração dos sistemas	Integração e coexistência passam a ser pré-requisitos
Não existência da Internet	Internet e comércio eletrônico é fundamental

Fonte: ([SES1998])

Objetos distribuídos é um paradigma da computação que permite a objetos serem distribuídos através de uma rede heterogênea, e permitir a cada componente a total interoperabilidade e acesso a outros objetos. Para uma aplicação escrita em um ambiente de objetos distribuídos, objetos remotos são acessados como se estivessem na máquina que efetua as requisições. Um objeto distribuído é essencialmente um componente, que pode interoperar com outros objetos distribuídos através de sistemas operacionais, redes, linguagens, aplicações, ferramentas e equipamentos diversos. Existe uma tendência de se construir sistemas computacionais abertos utilizando objetos distribuídos. Além disso, esses objetos distribuídos possuem as mesmas características principais dos objetos das linguagens de programação: encapsulamento, polimorfismo e herança, tendo, dessa forma, as mesmas principais vantagens: fácil reusabilidade, manutenção e depuração, só para citar algumas ([CAP1999]).

Mas objeto distribuído não opera sozinho. A princípio ele é construído para trabalhar com outros objetos e, para isso, precisa de uma espécie de barramento. Tais barramentos fornecem infra-estrutura para os objetos, adicionando novos serviços que podem ser herdados durante a construção do objeto, ou mesmo em tempo de execução para alcançar altos níveis de colaboração com outros objetos independentes ([CAP1999]).

Dois exemplos desses barramentos de tecnologias que habilitam a computação distribuída são o CORBA da OMG, que utiliza como *middleware* o ORB (*Object Request Broker*) e o DCOM da *Microsoft*®, que utiliza o COM (*Component Object Model*) como elemento que promove a comunicação entre um objeto-cliente e um objeto-servidor ([FRA1997],[CAP1999]). O CORBA especifica a arquitetura completa necessária à comunicação entre objetos distribuídos. Enquanto que a arquitetura DCOM, que encontra-se disponível no Windows 95, no Windows NT e Windows 2000, possui total controle dessas plataformas. Entretanto, conforme [HON1999], essa arquitetura apresenta deficiências no que diz respeito à sua utilização em outras plataformas.

## 2.9.2 ACTIVEX

Conforme [MIC2000], *ActiveX* é um conjunto de tecnologias que permitem construir aplicações empresariais para a Internet e intranets que executam em plataformas múltiplas (até bem pouco tempo atrás a tecnologia *ActiveX* ainda não era aceita em outros sistemas operacionais, a não ser pelo seu criador, o Windows 95, e pelo WindowsNT). Atualmente, aplicações *ActiveX* podem ser escritas em C, C++, Delphi e Java e podem ser executadas em plataformas como as da família Windows, Macintosh, e UNIX. Objetos Java e OLE atualmente suportam *ActiveX*. Por exemplo, pode-se criar uma aplicação para recuperar informações escrevendo um *applet* Java para a interface de usuário e usando funcionalidade de um componente de software existente que conecta com um banco de dados.

O que é agora conhecido como controle *ActiveX* foi chamado originalmente controle OLE e estes controles exigiram um número grande de métodos implementados. Isto os fez demasiadamente grandes. Eles estavam carregando e arrastando consigo tantas informações e características de interface que os componentes OLE não podiam viajar com rapidez por uma conexão típica da Internet. A partir daí foi construída uma variedade mais ágil e mais adaptável, que são os controles *ActiveX*. Estes podem implementar só as características que são absolutamente necessárias para o funcionamento dos controles, e nenhuma carga desnecessária foi requerida.

O *ActiveX* é, então, um novo nome para o novo enfoque, o do desenvolvimento para Internet, e engloba as tecnologias chave da *Microsoft*® de reutilização de funções programadas e fornecimento de serviços entre aplicativos COM e OLE.

Os controles *ActiveX* não podem executar por si só. Eles precisam de um recipiente. Inicialmente, este recipiente era o Visual Basic mas atualmente há muito mais escolhas. Um exemplo especialmente importante de um recipiente de controle *ActiveX* hoje é um *browser*.

Um engano terminológico muito comum, é confundir *ActiveX* com os controles *ActiveX*. O *ActiveX*, como já foi exposto anteriormente, é um conjunto de tecnologias e um controle *ActiveX* é um objeto COM que segue certos padrões ao interagir com seu cliente. Um controle *ActiveX* permite usar o mesmo controle em muitos contextos diferentes. Através de suas interfaces padronizadas, pode fazer virtualmente qualquer coisa e controla implementações de todos os tipos de funções, além de estar disponível em várias companhias de software atuais.

Do mesmo jeito que um controle *ActiveX* é usado num programa, ele poderá ser usado numa página *HTML*, sozinho, como um *applet* da linguagem *Java*. Dependendo da aplicação, esses controles podem ser usados por leigos ou só por programadores. Pode existir um controle que o usuário só precise escrever uma frase para fazer o que ele quer, enquanto outro que trate com banco de dados precise de programação mais específica.

Os controles *ActiveX* foram criados com a finalidade de combinar duas áreas distintas das tecnologias de informática: os controles personalizados e uma redescoberta das idéias básicas que regem a OLE e a programação orientada a objetos. Esses controles são os primeiros controles baseados em DCOM (*Distributed Component Object Model*) ([KAU1997]). Um bom motivo, ainda conforme [KAU1997] para utilizar *ActiveX*, é que esta tecnologia permite utilizar a OLE através da Web. Isto significa que pode ser usado um navegador Web como um *front end* para um processador de textos, planilha eletrônica, banco de dados e ainda vários outros aplicativos.

Segundo [BOR1999] a tecnologia *ActiveX* tem muita utilidade no desenvolvimento de aplicações pequenas e sem necessidade de configuração para ambiente distribuído na Web com ou sem arquitetura de banco de dados Cliente/Servidor multicamadas.

Com *ActiveX*, programadores e WEB *designers*, que são programadores que desenvolvem páginas para internet, podem criar conteúdo interativo com uma grande variedade de ferramentas e criando aplicativos mais atrativos e práticos.

### 3 DELPHI 5.0 - INOVAÇÕES DO AMBIENTE

O Delphi está provando que é uma ferramenta de desenvolvimento completa, capaz de agradar desde o pequeno desenvolvedor até a mais exigente corporação. Segundo [CAN2000], o Delphi 5.0 é a versão mais completa em recursos dentro da família do produto.

Já existiam, conforme [CAN2000], desde o Delphi 3.0, alguns recursos importantes que a tornaram uma ferramenta muito poderosa. São eles:

- a) RAD (*Rapid Application Development*) é um termo largamente utilizado hoje na área de ferramentas de desenvolvimento que significa basicamente "alta produtividade". No Delphi 5.0 estão à disposição mais de 140 componentes já prontos para serem utilizados, além de uma imensa gama de bibliotecas desenvolvidas por terceiros com as mais variadas utilidades. Existem também recursos de ambiente como *CodeInsight* e *CodeCompletion* que ajudam no aprendizado da ferramenta garantindo um menor tempo para se alcançarem os resultados desejados;
- b) desenvolvimento multicamadas: o Delphi implementa a arquitetura multicamadas baseado em um *middleware* chamado MIDAS que usa a estrutura de distribuição de objetos DCOM da *Microsoft*®. O Delphi tem um conjunto de componentes específicos para fornecer toda a funcionalidade de aplicações distribuídas e não é necessário um conhecimento profundo do DCOM para poder implementar multicamadas;
- c) programação orientada a objetos: com o Delphi, o desenvolvedor tem a capacidade de criar aplicações utilizando a metodologia de programação orientada a objetos;
- d) aplicações para servidores Web: permite a criação de aplicações que serão executadas em servidores Web. Isso, na verdade, não significa apenas a criação de formulários *ActiveX* para serem exibidos dentro de páginas Web, mas sim aplicações realmente voltadas para servidores. O que é importante ressaltar, nesse recurso, é a independência de padrões de servidores que essas aplicações possuem;

Mas o Delphi 5.0 apresenta alguns novos recursos, muito importantes, apresentados na tabela 4.

Tabela 4 - Novos Recursos do Delphi 5.0

Recurso	Descrição
Suporte a <i>Active Server Object Wizard</i> e ASP	ASP é um novo recurso do IIS ( <i>Internet Information Server</i> ), que fornece uma estrutura de aplicativo para desenvolvimento de funcionalidade mais poderosa em sites WEB. O Delphi 5.0 disponibiliza a utilização deste recurso com facilidade.
Componentes <i>DataSet</i> para ADO	No Delphi 5.0 estão disponíveis componentes ADO. Estes componentes são herdados da classe <i>TDataSet</i> e podem ser utilizados normalmente para apresentar e editar dados. Eles já implementam automaticamente a tecnologia ADO.
Servidores COM em Componentes	São componentes visuais, prontos para serem utilizados na programação distribuída através da tecnologia COM e DCOM. Atualmente já existem servidores COM em componentes prontos no ambiente Delphi.
Diagramas de Dados	O Delphi apresenta na versão 5.0 um diagrama de dados (faz parte do também novo <i>Data Module Design</i> ). O Diagrama de dados mostra uma visualização dos componentes, incluindo relacionamento mestre/detalhe, conexões de pesquisa, propriedades vinculadas e relacionamentos genéricos.
<i>Data Module Designer</i>	Desde a versão 2.0 o Delphi tem usado a idéia de módulo de dados, que é um contêiner de componentes relacionados aos dados da aplicação. No Delphi 5.0, o <i>Data Module Designer</i> permite selecionar componentes e conectá-los de maneira fácil, usando a visualização de árvore. Pode-se ver uma parte somente do aplicativo ou este como um todo.
<i>Internet Express</i>	É uma arquitetura que implementa a idéia de gerar página HTML na Web para permitir que qualquer usuário interaja com o servidor da camada do meio, através de um servidor Web. Seria uma arquitetura de quatro camadas: O servidor SQL, o servidor do aplicativo (pode ser um servidor MIDAS), o servidor Web e finalmente, o navegador da Web.
MIDAS 3	O MIDAS é um conjunto de tecnologias distintas que trabalham juntas para tornar mais fácil construir aplicativos distribuídos com Delphi. No MIDAS 3 existe apenas uma interface de comunicação entre os aplicativos, enquanto nas versões anteriores existiam duas.
<i>Integrated Translation Environment</i> (ITE)	Um dos mais novos recursos do Delphi é o ITE, que possibilita a tradução de títulos de telas, descrições, etc, para outras línguas através do <i>Translation Manager</i> .

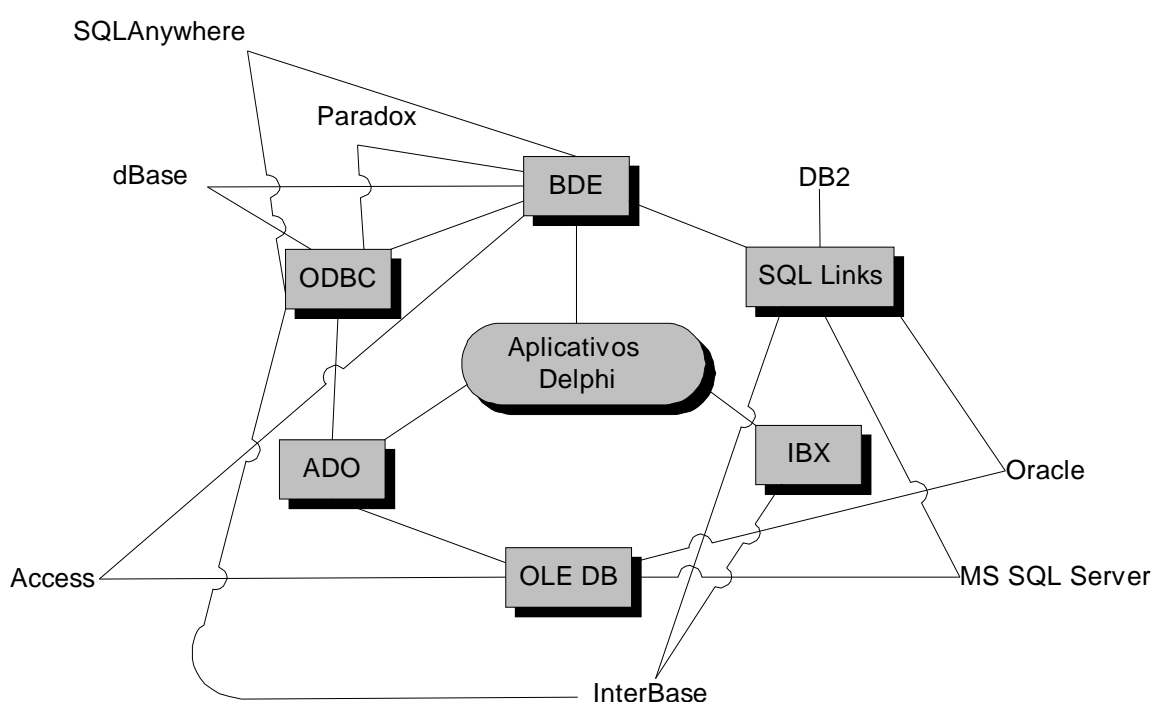
Fonte : ([CAN2000])

### 3.1 BANCO DE DADOS NO DELPHI 5.0

Os aplicativos de banco de dados Delphi não têm acesso direto às fontes de dados que fazem referência e não podem manipular arquivos de bancos de dados diretamente. Em vez disso, eles usam um mecanismo de bancos de dados disponível, como o *Borland Database Engine* (BDE) ou *ActiveX Data Objects* (ADO).

Através da Figura 7, pode-se verificar como as tecnologias de acesso a dados alternativas são disponibilizadas no Delphi 5.0.

**Figura 7 - As tecnologias de acesso a dados alternativas disponíveis no Delphi 5.0**



Fonte: ([CAN2000])

O BDE tem acesso direto a várias fontes de dados, incluindo dBase, Paradox, ASCII, FoxPro, a até tabelas Access. O BDE também pode fazer interface com o SQL Links da Borland, que consistem em uma série de *drivers* que permitem acesso a vários servidores SQL locais e remotos. Os servidores de banco de dados incluem Oracle, Sybase, Informix, InterBase, SqlAnywhere, DB2, etc. Se for necessário fazer acesso com alguma base de dados ou algum formato de dados diferente, o BDE pode fazer a interface com os drivers ODBC, embora, neste caso, possa ser utilizado também o ADO. Os aplicativos construídos com os componentes ADO do Delphi 5.0, não exigem as bibliotecas BDE ([CAN2000]).



O IBX (*InterBase Express*) apresenta novos componentes que existem no Delphi 5.0 projetados especificamente para acessar um banco de dados InterBase, sem precisar da BDE.

### 3.2 DELPHI 5.0 E ADO

Quando o Delphi foi projetado originalmente, a Borland possuía o Paradox e o Visual dBase e, portanto, construiu um único mecanismo de acesso a dados compartilhado por todos os seus produtos. Esse mecanismo é o BDE. O BDE foi projetado para muitos servidores SQL, incluindo Oracle e InterBase, e para acessar outra origem de dados através de ODBC. O ODBC, entretanto, era originalmente muito lento e alguns dos *drivers* para formatos de bancos de dados específicos não estavam livres de erro. Com o passar do tempo, três coisas aconteceram: primeiro, a Borland vendeu seus produtos de banco de dados de usuário final; segundo, o papel da *Microsoft*® como fornecedora de banco de dados ficou maior (com o MS Access e o SQL Server ganhando cada vez mais aceitação no setor), aumentando a frequência dos programas em trabalhar com banco de dados *Microsoft*®; terceiro, a *Microsoft*® aprimorou sua estratégia de acesso a dados, introduzindo RDO, OLE DB e ADO ([CAN2000]).

A Borland reconheceu estas alterações nos últimos anos, incluindo *drivers* específicos para o MS Access. Usando o BDE é possível conectar-se a um banco de dados Access, mas isso implica em usar dois mecanismos de banco de dados simultaneamente quando uma conexão do Access através do ADO é um vínculo mais direto. O Delphi 5.0 oferece suporte à tecnologia ADO diretamente, através de componentes *DataSet* específicos.

Na verdade, podem ser escritos programas ADO sem os componentes específicos que o Delphi 5.0 fornece. Entretanto, os componentes ADO de conjunto de dados prontos para usar permitem que o programador implemente o ADO utilizando as mesmas técnicas com as quais está habituado.

## 4 DESENVOLVIMENTO DO PROTÓTIPO

### 4.1 O TUTOR INTELIGENTE PARA O AMBIENTE DELPHI

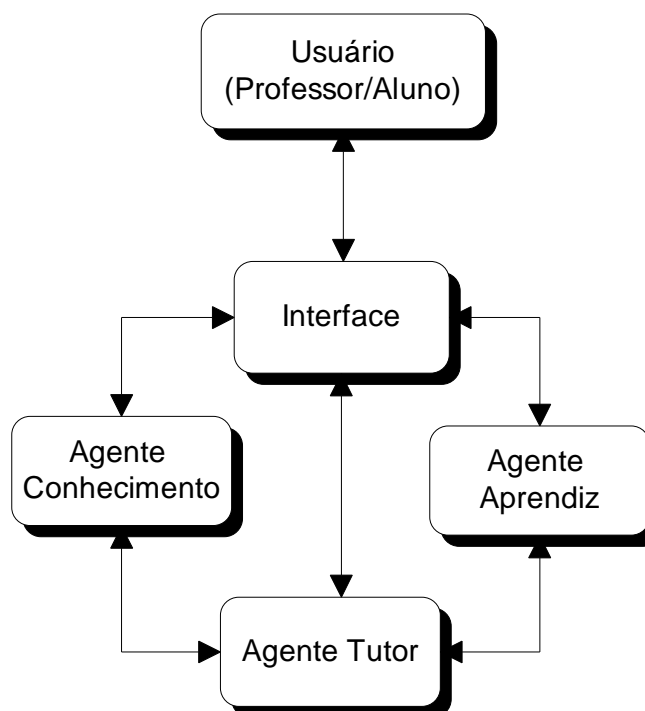
O desenvolvimento do protótipo relacionado a este trabalho não inicia da estaca zero. Ele é uma adaptação do Tutor Delphi, o qual foi implementado anteriormente, em outro Trabalho de Conclusão de Curso, do curso de Bacharelado em Ciências da Computação, abordado em [FRA1999].

Basicamente, o Tutor Delphi é formado por três etapas principais:

- a) cadastro de um exercício de programação em Delphi, o qual deve conter todas as informações necessárias para a execução do exercício, como por exemplo o objetivo do exercício, os componentes que deverão ser utilizados, os eventos associados a cada componente e a descrição da ação que cada evento deverá disparar. Somente o professor poderá cadastrar exercícios;
- b) a aprendizagem do exercício, através da qual o aluno vai realizar o exercício com ajuda de um agente, que irá auxiliá-lo, passo a passo, na execução do exercício. Esta etapa deve ser executada pelo aluno;
- c) a correção do exercício, que é realizada pelo tutor, o qual vai corrigir um determinado exercício do aluno que está requisitando a correção. Tanto o professor quanto o aluno podem fazer as correções dos exercícios, mas se for um aluno que solicitar a correção, somente será permitido fazer correções de exercícios feitos pelo próprio aluno. No caso do professor solicitar correção de exercícios, ele poderá escolher qual o exercício e de qual aluno pertence o exercício que será corrigido.

Cada uma dessas etapas corresponde a um objeto, ou ainda, um agente. Ao cadastro dos exercícios, está ligado o objeto conhecimento. Esse objeto mantém o conhecimento do exercício. A etapa de aprendizagem, corresponde ao objeto aprendiz. É onde o conhecimento é repassado ao aluno. A terceira etapa, a de correção do exercício, corresponde ao objeto tutor, que faz a correção do exercício realizado pelo aluno, armazena os erros e acertos, informando ao aluno e ao professor os resultados.

Na figura 8, pode ser verificada a relação entre os objetos (agentes) do sistema.

**Figura 8 - Relação entre os agentes**

Primeiramente o usuário interage com a interface do sistema. O usuário identifica-se e após obter o acesso, poderá iniciar a execução dos agentes.

Para maiores e mais detalhadas informações sobre o conceito de agentes inteligentes e também sobre o sistema desenvolvido pode ser consultado [FRA1999].

## 4.2 ESPECIFICAÇÃO DO PROTÓTIPO

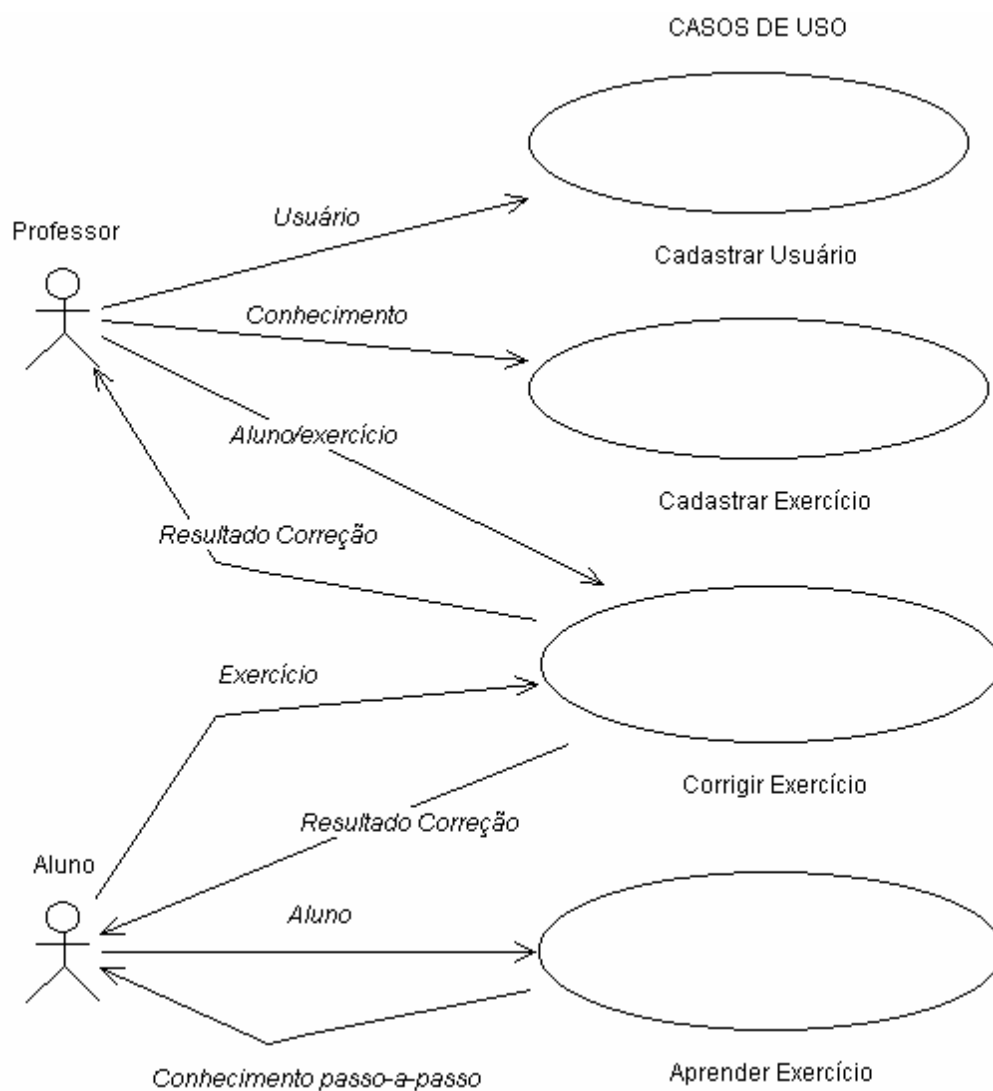
Como este protótipo é uma adaptação, a especificação foi revisada e foram feitas as devidas alterações, necessárias para a adequação com a atual versão.

Partindo-se da premissa de que a UML é uma linguagem de modelagem e não uma metodologia [FUR1998], então será demonstrada a modelagem do protótipo através da linguagem UML, utilizando-se uma metodologia de programação orientada a objetos. Para a modelagem, foram utilizados os diagramas de casos de uso, de classes e de seqüência, todos feitos na ferramenta de modelagem *Rational Rose* da *Rational*®.

## 4.2.1 CASOS DE USO

Algumas alterações no Tutor Delphi, foram realizadas, para melhor adaptá-lo ao propósito deste trabalho. Ao longo da especificação, serão destacadas as principais mudanças.

**Figura 9 – Diagrama de Casos de Uso**



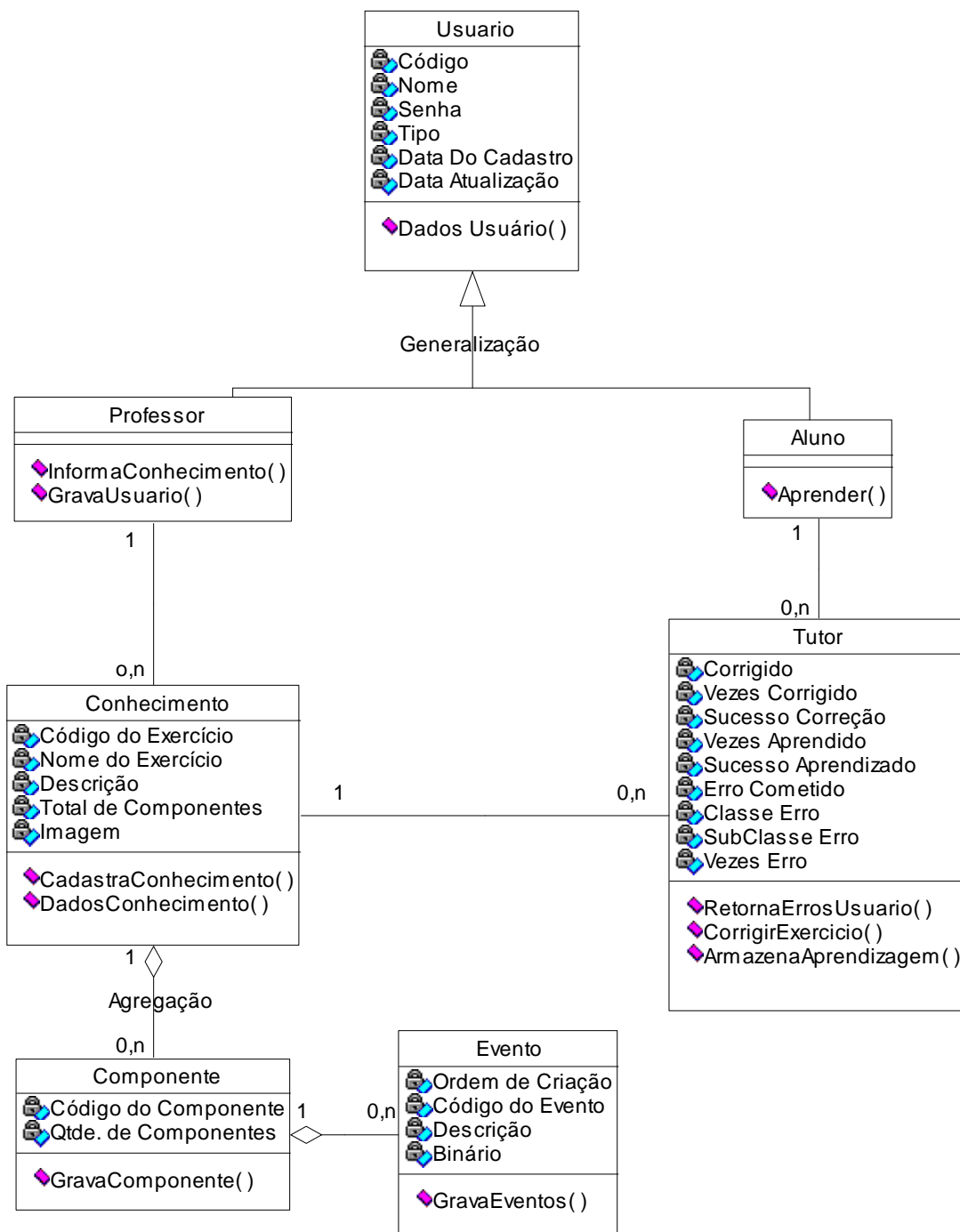
São 4(quatro) os casos de uso identificados neste programa:

- a) cadastrar Usuário: responsável pela entradas dos dados referentes aos usuários do sistema e a gravação dos mesmos na base de dados. Somente um usuário do tipo Professor pode cadastrar usuários. Mudança: Este caso de uso não existia na versão anterior. Naquela, o cadastro de usuário era livre, ou seja, qualquer aluno poderia ser cadastrado por qualquer usuário, no momento da execução do exercício. Não havia restrições em nível de usuário, pois a classe usuário era utilizada apenas para fazer o relacionamento com um determinado exercício;
- b) cadastrar Exercício: é a entrada dos dados referentes aos exercícios e a gravação dos mesmos nas entidades referentes ao conhecimento. Mudança: na versão anterior as informações sobre o conhecimento ficavam gravadas em arquivos .INI e na versão atual, foram criadas novas tabelas de dados para armazenar estas informações. Somente o professor poderá cadastrar exercícios;
- c) aprender Exercício: neste caso de uso, o aprendiz recebe o conhecimento e executa o exercício, informando seus dados. Mudança: neste passo não houve grandes alterações. O ponto principal é que a origem do conhecimento não é mais os arquivos .INI e sim as classes que armazenam o conhecimento;
- d) corrigir Exercício: o usuário identificado pelo sistema, escolhe um exercício e solicita a correção do mesmo. O sistema faz a correção e informa o resultado. Mudança: na primeira versão, havia duas classes (“AprenderExercício” e “ErrosCometidos”) que armazenavam o resultado do exercício. Nesta nova versão, há um objeto Tutor, que gerencia a correção dos exercícios e retorna os resultados desta correção para o professor ou aluno que solicitou a correção. Foi criada uma classe única para armazenar os resultados dos exercícios.

## 4.2.2 DIAGRAMA DE CLASSES

A figura 10 demonstra o Diagrama de Classes.

**Figura 10 - Diagrama de Classes**



Existem 7 (sete) classes identificadas no sistema:

- a) usuário: corresponde aos usuários cadastrados no sistema e controla os acessos dos tipos de usuários (professor e aluno). Mudança: nesta versão a classe tem especializações por tipo de usuário;
- b) professor: é uma especialização da classe usuário. Os atributos da classe professor são idênticos aos da classe usuário, porém o método “InformaConhecimento” é inerente somente a esta classe. Mudança: não existia;
- c) aluno: também é uma especialização da classe usuário, e diferencia-se da classe professor, pois implementa o método “Aprender”. Ela identifica um tipo de usuário (aluno) com permissões restritas no sistema. Mudança: não existia;
- d) conhecimento: é a classe que mantém o conhecimento que o professor repassou, e que irá fornecer este mesmo conhecimento para o agente aprendiz, passo a passo, e também, fornecer as informações sobre o conhecimento, necessárias à correção do exercício. Mudança: não houve;
- e) componente: esta classe é uma agregação da classe conhecimento. Um conhecimento, ou seja, um exercício, poderá conter vários componentes. A classe componente mantém o controle dos componentes referentes a um conhecimento. Mudança: agrega eventos;
- f) evento: assim como os componentes são agregações da classe de conhecimento, os eventos são uma agregação da classe de componentes. Cada componente pode ter vários eventos associados a ele. A classe evento controla o cadastro dos eventos dos componentes de um exercício, dentro das limitações do sistema (alguns eventos apenas são tratados). Mudança: não houve;
- g) tutor: a classe tutor é responsável pela associação entre o conhecimento e o aprendiz, controlando a troca de informações entre as mesmas. Esta classe gerencia a correção dos exercícios. Mudança: contém os atributos resultantes da correção de um exercício/conhecimento.

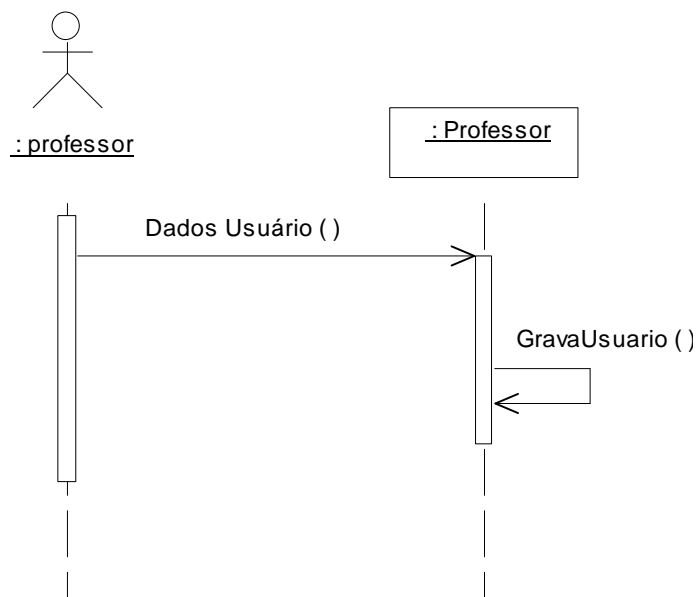
## 4.2.3 DIAGRAMAS DE SEQÜÊNCIA

Os diagramas de seqüências representam, como o próprio nome indica, a seqüência em que as ações ocorrem dentro do sistema. Eles demonstram como é feita a troca de mensagens entre as classes. Para cada caso de uso, há um diagrama de seqüência, conforme as figuras 11, 12, 13 e 14.

### 4.2.3.1 CADASTRAR USUÁRIO

O ator do tipo professor informa os dados do usuário, que são gravados através da rotina “GravaUsuario” da classe professor.

**Figura 11 - Diagrama de seqüência Cadastrar Usuário**

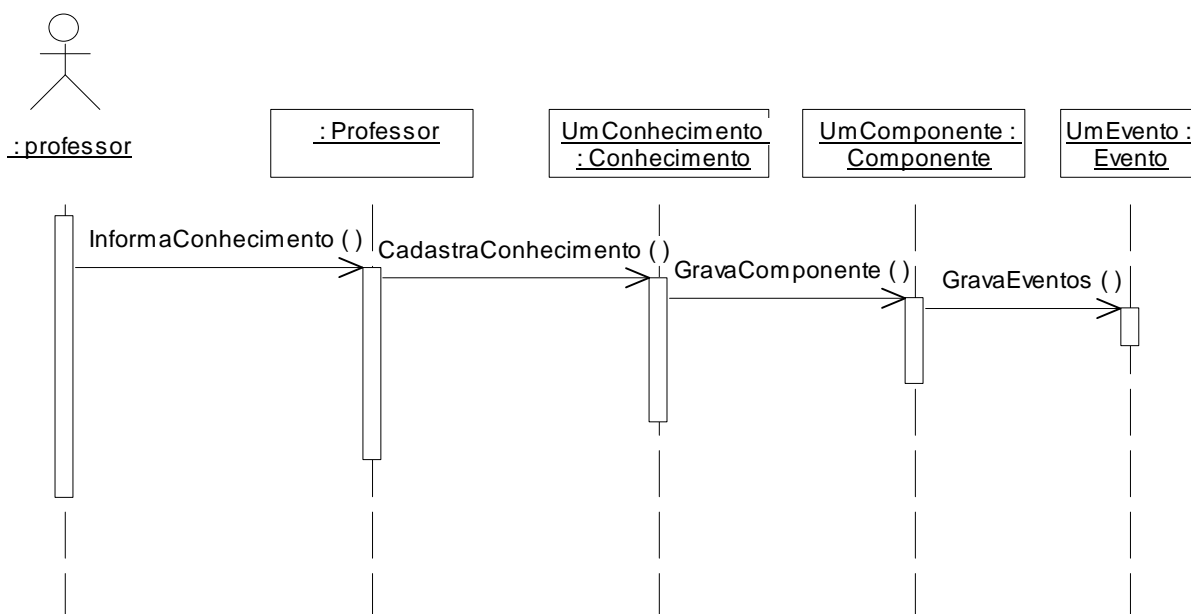




### 4.2.3.2 CADASTRAR EXERCÍCIO

O ator do tipo professor informa o conhecimento para a classe professor, depois este conhecimento é cadastrado através do método “CadastraConhecimento” da classe conhecimento, que por sua vez grava os componentes e os eventos.

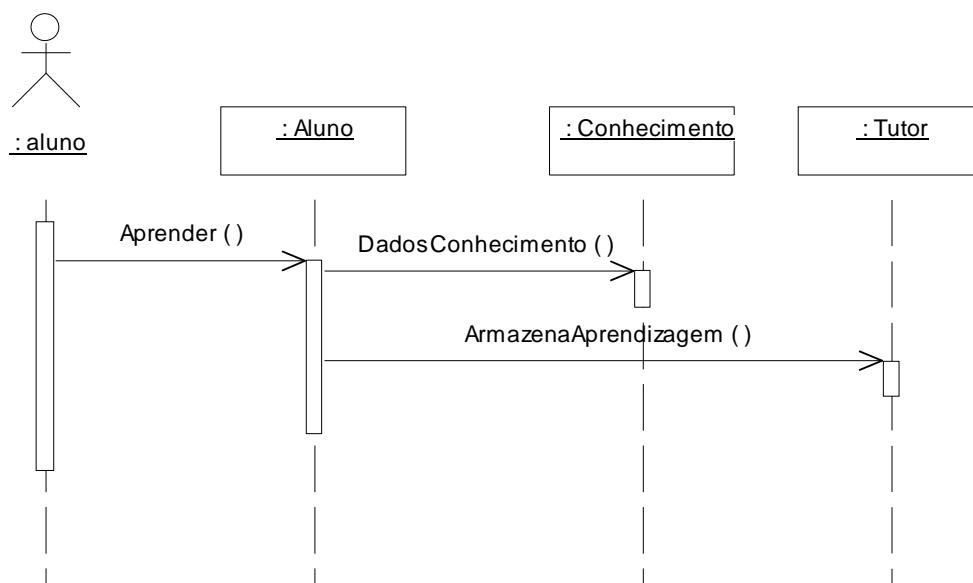
Figura 12 - Diagrama de seqüência Cadastrar Exercício



### 4.2.3.3 APRENDER EXERCÍCIO

O ator aluno inicia o processo de aprendizagem, informando para os dados do conhecimento, ou exercício, que irá aprender, e a aprendizagem é armazenada, através do método “ArmazenaAprendizagem”, da classe tutor.

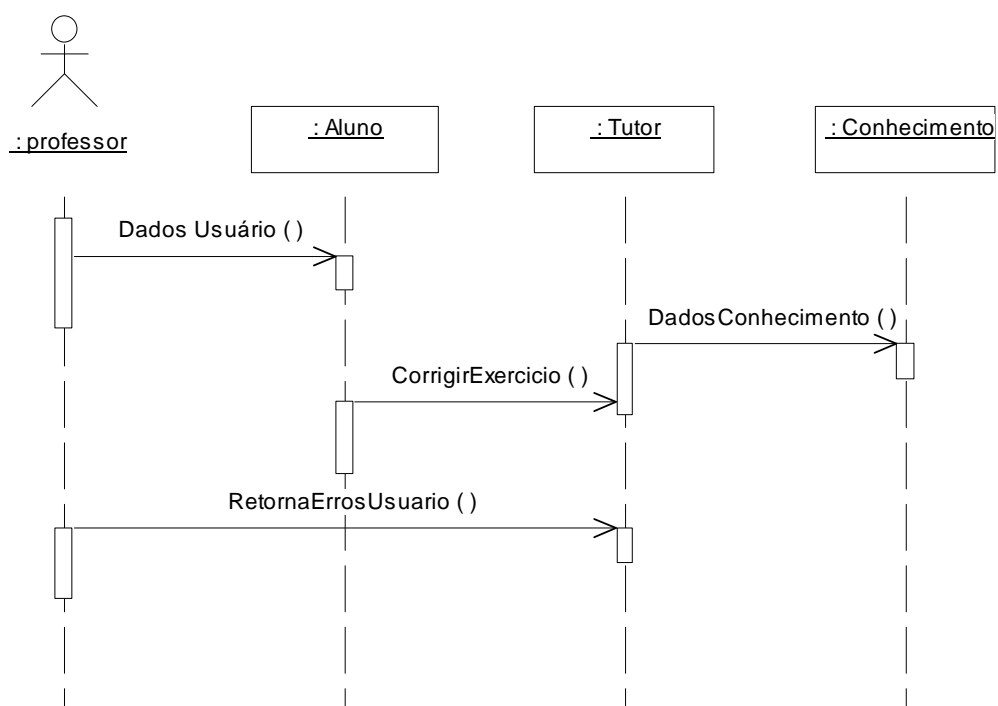
**Figura 13 - Diagrama de seqüência Aprender Exercício**



#### 4.2.3.4 CORRIGIR EXERCÍCIO

Este processo pode ser iniciado tanto pelo ator professor quanto pelo ator aluno. Se for o professor, ele informa os dados do aluno, solicita a correção do exercício, que por sua vez vai buscar os dados do conhecimento através do método “DadosConhecimento” da classe conhecimento, e após, é retornado o resultado da correção para o professor.

**Figura 14 - Diagrama de seqüência Corrigir Exercício**



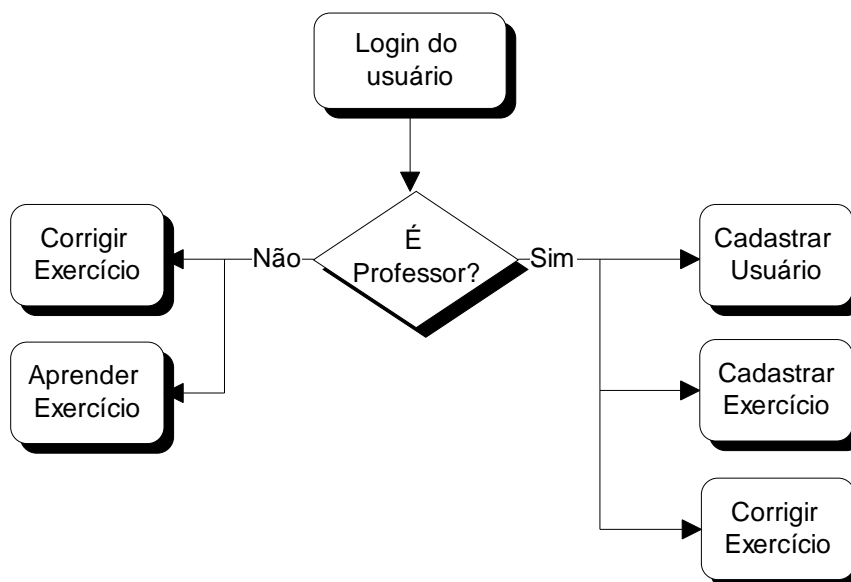
Se o processo for iniciado por um ator do tipo aluno, a principal diferença é que não será preciso informar os dados do aluno, pois eles são os do próprio aluno que está solicitando a correção do exercício.

As mudanças destes diagramas de seqüências, em relação aos da versão anterior do Tutor Delphi, são oriundas das mudanças já destacadas no capítulo 4.2.1, referentes aos casos de usos, e das mudanças destacadas no capítulo 4.2.2, referentes às alterações nas classes utilizadas pelo sistema.

### 4.3 DEMAIS MUDANÇAS NO TUTOR DELPHI

Em decorrência da criação das especializações da classe usuário, foram realizadas algumas mudanças no Tutor Delphi, também com relação às permissões do usuários. Após sua conexão com o sistema, o usuário poderá interagir com qualquer um dos agentes, através da interface, desde que tenha permissão para tal. As permissões são diferenciadas para usuários do tipo Aluno ou do tipo Professor, conforme demonstrado na figura 15.

**Figura 15 - Permissões do usuário**



Os usuários podem ser cadastrados somente por professores. No cadastro do usuário, é identificado o tipo de usuário, que pode ser aluno ou professor. Ao aluno, somente é dada permissão para aprender/resolver algum exercício (qualquer um que esteja cadastrado na base de conhecimento) e corrigir somente os seus exercícios.

Aos usuários do tipo professor será permitido fazer inclusão de usuários, cadastro de exercícios e correção de exercícios de qualquer aluno.

Algumas outras mudanças básicas que foram feitas no programa, estão relacionadas na tabela 5:

**Tabela 5 - Mudanças básicas no Tutor Delphi**

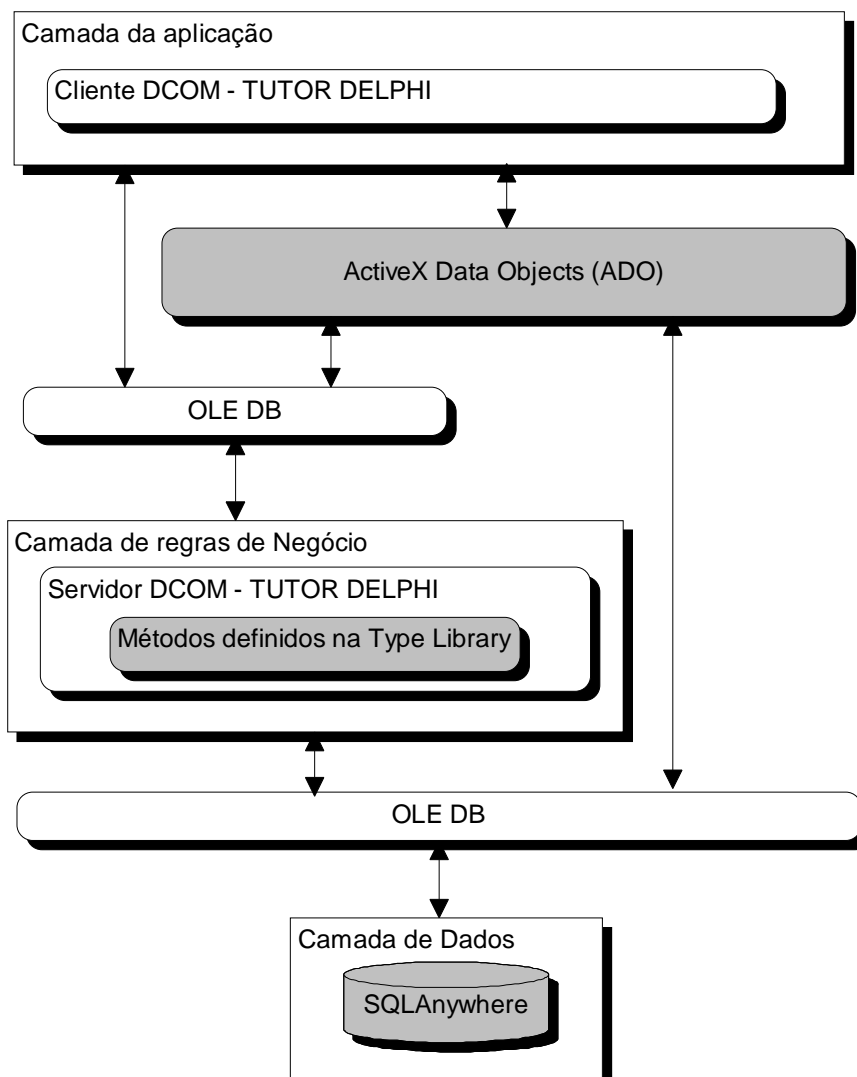
<b>Assunto</b>	<b>Versão Antiga</b>	<b>Versão Atual</b>
Execução	<i>Standalone</i> (cada agente era executado separada e independentemente).	Cliente/Servidor (cada agente é um objeto, chamado através da interface, que está no programa cliente. O Cliente e o Servidor comunicam-se remotamente através de DCOM).
Informações	Informações armazenadas parte em tabelas Paradox e parte em arquivos .INI, mantidos na própria estação.	As informações são todas gravadas em tabelas. Optou-se por utilizar para demonstração o banco de dados Sybase SQLAnywhere.
Acesso aos dados	Através de componente TTable do Delphi.	Através de componentes ADOTable, encapsulando toda a tecnologia ADO.
Conhecimento	Todo o conhecimento adquirido ficava armazenado nos arquivos .INI na estação local.	O conhecimento fica disponível na base de dados do servidor, na camada de dados.
Segurança	Qualquer aluno pode cadastrar outros usuários.	Somente professores podem cadastrar novos usuários.

## 4.4 O TUTOR DELPHI NUMA VISÃO DISTRIBUÍDA

A nova versão do Tutor Delphi foi modelado baseado na arquitetura de três camadas.

A figura 16 mostra a distribuição dos componentes entre as camadas do Tutor Delphi.

**Figura 16 - Camadas do Tutor Delphi**



As camadas podem estar fisicamente separadas, cada qual em uma máquina diferente, ou todas em uma única máquina. O programa cliente implementa a camada da aplicação e o programa servidor, a camada de regras de negócios. A camada de dados também independe da localização.

A comunicação entre as camadas se dá através da tecnologia que implementa objetos distribuídos, chamada DCOM. Atualmente existem várias maneiras para se implementar objetos distribuídos. Como o protótipo do Tutor Delphi foi implementado em Delphi, foram utilizados componentes do Delphi 5.0 que disponibilizam os recursos necessários para a implementação de um programa cliente e um programa servidor, comunicando-se remotamente através de DCOM.

DCOM é uma extensão do COM. Quando cliente e servidor residem em diferentes máquinas, DCOM substitui a comunicação inter-processos local por um protocolo de rede. No trabalho em questão utilizou-se o DCOM, devido à maior compatibilidade que esta tecnologia oferece em aplicações Windows e também pelo fato do Delphi ter total suporte e componentes para sua utilização.

DCOM é a tecnologia que a *Microsoft*® vem tentando difundir em computadores que rodam seu Sistema Operacional, tanto que não é necessário qualquer outro produto para que seja possível a comunicação entre objetos distintos, ou seja, o Windows já está preparado para a utilização de DCOM ([CAP1999]).

Clientes DCOM podem requisitar métodos em objetos que estão fora do processo atual, na mesma máquina, ou em um processo remoto em alguma máquina na rede. Quando um cliente chama um método que está localizado no mesmo processo, esse método é acessado diretamente([CAP1999]). O modelo DCOM tem a característica da transparência de localização, que simplifica bastante a tarefa de distribuição de componentes para otimização de performance.

Para que os objetos entendam-se e possam trabalhar juntos, é necessário que tenham interfaces comuns, ou pelo menos que possam conhecer as interfaces dos outros objetos. O modelo de objetos define uma linguagem de definição de interfaces, a *Interface Definition Language* (IDL). Esta especificação é independente da linguagem de programação, sistema operacional ou arquitetura. A IDL define os tipos de objetos, seus atributos (propriedades), as funções que expõem (métodos), os parâmetros a serem recebidos no caso de uma mensagem recebida, e possíveis exceções a serem tratadas ([CAP1999]).

Utilizando-se os recursos do Delphi 5.0, a IDL será encapsulada e gerada automaticamente pela *Type Library* (Biblioteca de tipos).

A *Type Library* é um arquivo construído automaticamente pelo Delphi 5.0, quando se cria um servidor remoto, ou é criado pelo *Library Editor* do Delphi, para aplicativos de servidor OLE. Ela é um conjunto de informações de tipo que descreve todos os elementos (objetos, interfaces...) disponibilizadas por um servidor. A principal diferença entre uma *TypeLibrary* e outras descrições destes elementos, como código pascal, por exemplo, é que

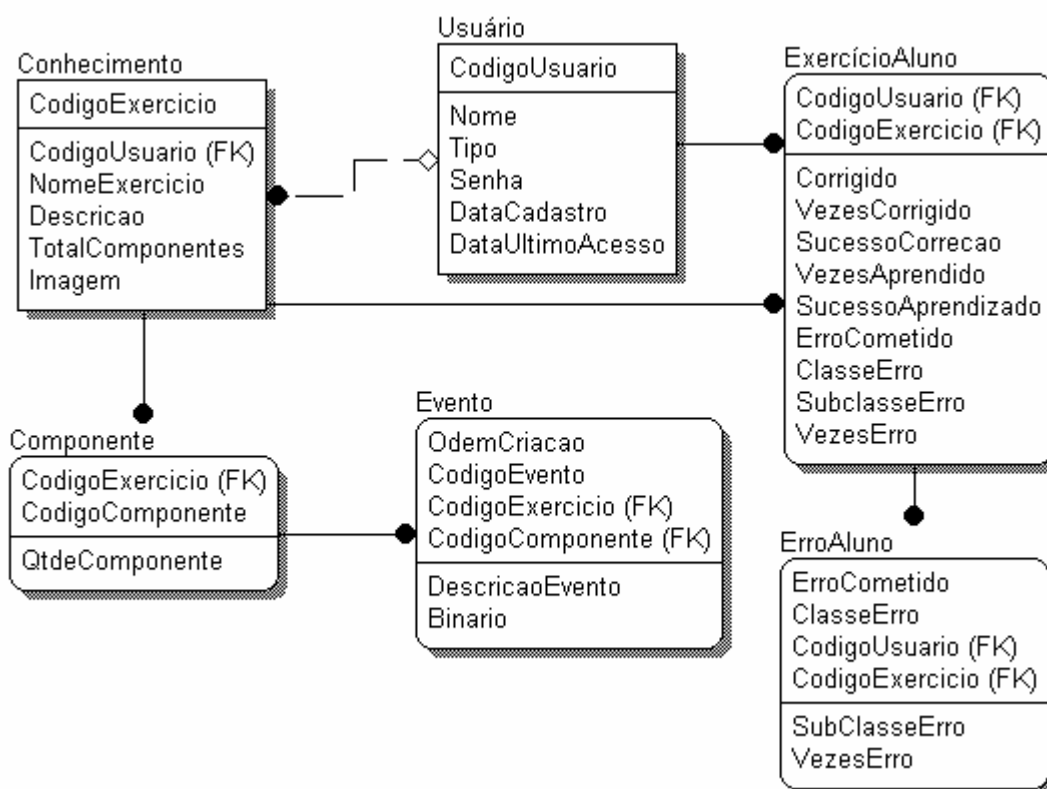
uma *TypeLibrary* é independente de linguagem. Os elementos de tipo são definidos pelo OLE como um subconjunto dos elementos padrão das linguagens de programação e qualquer ferramenta de desenvolvimento, segundo [CAN2000], pode utilizá-los.

#### 4.4.1 A CAMADA DE DADOS

Sobre a camada de dados não há muito o que descrever. Nesta camada ficam os dados utilizados no protótipo. Se resume basicamente em um sistema gerenciador de bancos de dados e um arquivo de dados, com as entidades do sistema e os dados gravados até o momento. O banco de dados utilizado no Tutor Delphi é o SQL Anywhere.

No Tutor Delphi, existem 5 entidades, conforme o modelo de dados demonstrado na figura 17:

**Figura 17 - Modelo de Dados**

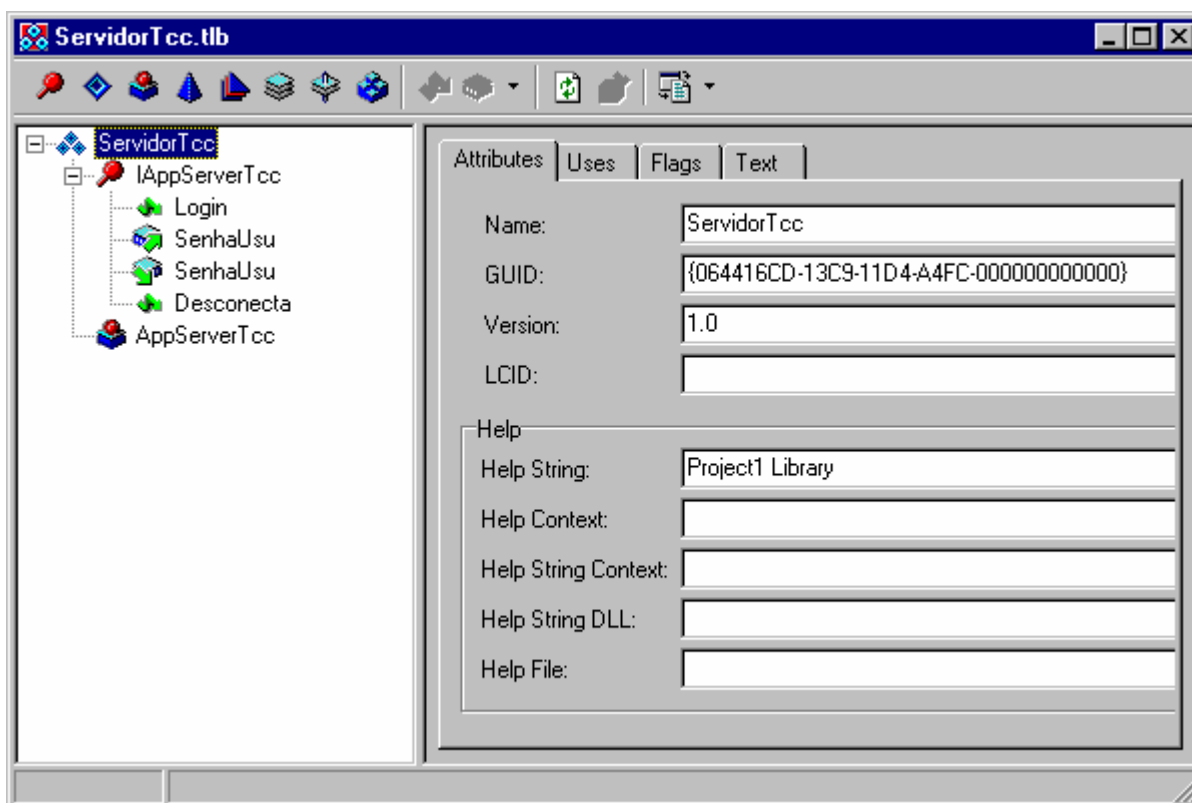


## 4.4.2 A CAMADA DE REGRAS DE NEGÓCIOS

Para a implementação da camada de regras de negócios do Tutor Delphi, foi criado um novo aplicativo no Delphi 5.0, incluindo nele um módulo de dados remoto. Através do *Remote Data Module Wizard*, disponível no ambiente, deve ser informado o nome da classe do servidor, neste caso denominada “AppServerTcc”. Então, o Delphi coloca um módulo de dados no programa, que terá propriedades e eventos normais, mas a sua classe será definida na biblioteca de tipos (*Type Library*), que é criada automaticamente. É nesta *Type Library* que serão definidos todos os métodos e propriedades utilizadas no nível intermediário.

Através do Editor *Type Library* demonstrado na figura 18, pode-se verificar que a classe “ServidorTcc” implementa uma interface específica (interface *IUnknown*), chamada “IAppServerTcc”, a qual implementa um método “Login”, um método “Desconecta” e uma propriedade “SenhaUsu”.

**Figura 18 - O Editor Type Library**





Ao criar-se os métodos na *Type Library*, o Delphi faz as declarações necessárias para que a implementação do método seja escrita pelo programador. O quadro 1 apresenta a definição dos métodos da interface *IappServerTcc*.

### Quadro 1 - Definição dos Métodos da Type Library

```
procedure Login(Codigo: Integer; const Senha: WideString);  
safecall;  
function Get_SenhaUsu: WideString; safecall;  
procedure Set_SenhaUsu(const Value: WideString); safecall;  
procedure Desconecta(CodUsu: Integer); safecall;
```

O método *Login* recebe o código e a senha como parâmetros, verifica se a senha está correta e permite ou não o acesso. A convenção de chamada *safecall* permite ao Delphi manipular automaticamente os códigos de erro ([CAN2000]).

O método *Desconecta* apenas atualiza a tela de *logs* do servidor. O parâmetro indica qual usuário desconectou-se.

A propriedade *SenhaUsu* utiliza os métodos *Get\_SenhaUsu* para leitura e *Set\_SenhaUsu* para gravação. Estes métodos são responsáveis, respectivamente, por informar a senha do usuário conectado (já criptografada) e por atualizar a senha, criptografando-a, quando o usuário é criado ou quando está se conectando ao sistema.

No anexo II estão os códigos fontes da *Type Library* e também da implementação dos métodos e propriedades definidos nela. Esta implementação, propriamente dita, encontra-se na classe *TAppServerTcc*, que é criada no formulário correspondente ao Módulo de Dados Remoto do *ServidorTcc*, que por sua vez, encontra-se na camada de regras de negócio.

No que se refere à comunicação entre as camadas, a tecnologia COM define um modo padronizado para um módulo cliente e módulo servidor se comunicarem entre si através de uma interface específica. Os dois módulos podem ser executados no mesmo computador ou através de uma rede. Muitas interfaces são possíveis, dependendo do papel do cliente e do servidor, e pode-se incluir novas interfaces para propósitos específicos. Estas interfaces são implementadas por objetos servidor. Um objeto servidor normalmente implementa mais de uma interface, e todos os objetos servidor têm alguns recursos comuns, pois todos eles devem implementar a classe *IUnknown*, que é a interface base a partir da qual são geradas as interfaces específicas de cada programa. O Delphi fornece uma classe com implementações

prontas para serem utilizadas com a classe *IUnknown*, chamada *TComObject* que é utilizada para exportar um objeto de um servidor. Várias outras classes herdam *TComObject* e oferecem suporte a mais interfaces, que são exigidas pelos controle *ActiveX*, por exemplo ([CAN2000]).

A interface *IUnknown* tem três métodos: *Add*, *Release* e *QueryInterface*. No quadro 2 pode ser vista a definição da interface (extraída da unidade *System* do Delphi):

### Quadro 2 - Definição da interface *IUnknown*

```
IUnknown = interface
  ['{00000000-0000-0000-C000-000000000046}']
  function QueryInterface(const IID: TGUID; out Obj):
    HRESULT; stdcall;
  function _AddRef: Integer; stdcall;
  function _Release: Integer; stdcall;
end;
```

Os métodos *\_AddRef* e *\_Release* são usados para implementar contagem de referência e o método *QueryInterface* manipula as informações e a compatibilidade de tipos de objetos. Normalmente não é preciso implementar esses métodos, pois eles podem ser herdados da uma das classes do Delphi que já oferecem suporte a eles.

No quadro 3 está a criação do objeto COM referente ao servidor do Tutor Delphi. O código foi gerado automaticamente na classe *AppServerTcc*, quando criou-se o objeto *ServidorTcc*.

### Quadro 3 - Criação da interface *Servidor Tutor Delphi*

```
type
  IAppServerTcc = interface;
  AppServerTcc = IAppServerTcc;
end;
uses ComObj;
class function CoAppServerTcc.Create: IAppServerTcc;
begin
  Result := CreateComObject(CLASS_AppServerTcc) as IAppServerTcc;
end;
class function CoAppServerTcc.CreateRemote(const MachineName: string):
  IAppServerTcc;
begin
  Result := CreateRemoteComObject(MachineName, CLASS_AppServerTcc) as
  IAppServerTcc;
```

Nota-se que há dois métodos para criação do objeto COM: o *Create* e o *CreateRemote*. A diferença entre eles é que a criação de um objeto remoto necessita do nome da máquina como parâmetro para sua criação.

É importante notar também na chamada *CreateComObject* a conversão *as*. A Chamada da API inicia o mecanismo de construção do objeto COM e o valor de retorno é um objeto *IUnknown*. Como mencionado anteriormente, este objeto é então convertido para o tipo de interface correto, antes que se inicie sua utilização. Neste caso o tipo de interface é a interface do servidor “IAppServerTcc”.

Outro atributo da classe “ServidorTcc” é o GUID (*Globally Unique Identifiers* ou identificador globalmente exclusivo) (ver figura 18). Ele é uma constante do tipo TGUID utilizada no método *QueryInterface*. Esse é um código que identifica qualquer classe de servidor COM/DCOM e qualquer interface no sistema. Um GUID é gerado quando o programa faz uma chamada da API *CoCreateGuid*, através de seu ambiente de desenvolvimento.

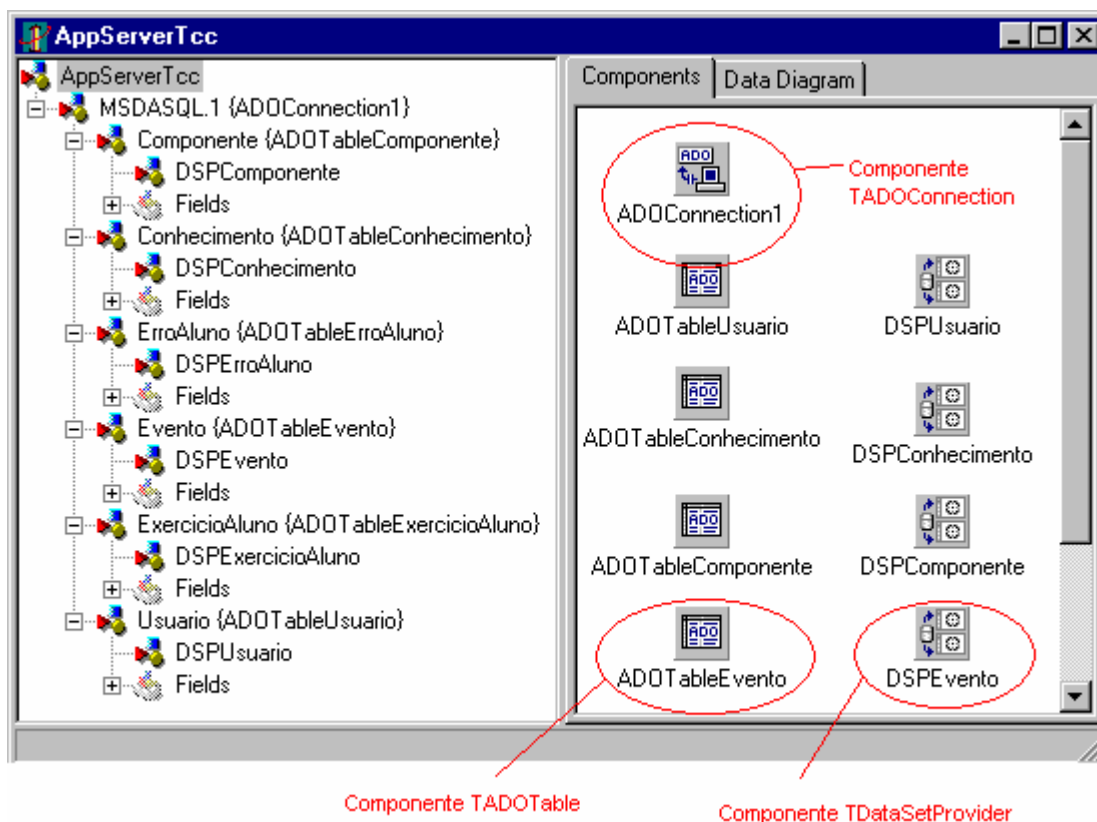
Segundo [CAN2000], mesmo sendo o GUID um número com 38 dígitos, deve-se tomar cuidado para não copiar um GUID de algum outro programa, o que resultaria em dois objetos COM completamente diferente utilizando o mesmo GUID. O Delphi possui um recurso para evitar que o programador crie seu GUID manualmente, introduzindo uma seqüência casual de números: basta digitar Ctrl+Shift+G no editor do Delphi, e será obtido um novo GUID corretamente definido.

#### **4.4.2.1 COMPONENTES BÁSICOS DO SERVIDOR**

Neste ponto, tem-se um Módulo de Dados Remoto vazio, que foi criado no *Remote Data Module Wizard*, ao ser iniciado o projeto, juntamente com a *Type Library*. É preciso informar, então, onde estão os dados necessários para a aplicação e como eles serão acessados.

A figura 19 demonstra o Módulo de Dados do ServidorTcc, onde estão os componentes que fazem os acessos aos dados do sistema.

Figura 19 - Módulo de Dados Remoto do ServidorTCC

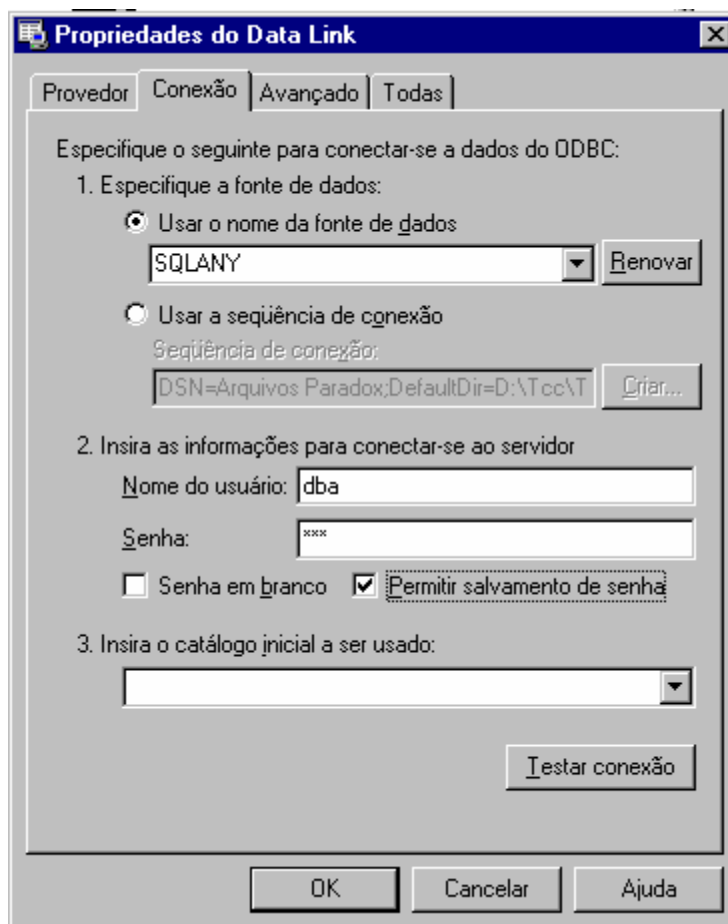


No Tutor Delphi, os componentes de acesso aos dados no lado do cliente e no lado servidor, comunicam-se via DCOM, e estão ligados aos dados através de componentes ADO. Os componentes ADO do Delphi 5.0 encapsulam toda a tecnologia descrita anteriormente neste trabalho.

No lado do servidor, componentes TADOTable, que são derivados do TTable, estão ligados às tabelas de dados através de uma conexão ADO. A Conexão ADO (feita através do componente *TADOConnection*, pois cada componente *ADOTable*, está ligado ao *ADOConnection* através da propriedade *Connection*) encapsula o objeto *ADOConnection*. Uma conexão ADO pode prover múltiplos conjuntos de dados (*ADODataset*) e comandos através de suas propriedades. A conexão permite ao programador o total controle dos atributos e condições sobre a base de dados conectada.

O componente *TADOConnection*, tem uma propriedade chamada *ConnectionString*, a qual deve ser configurada de acordo com a fonte de dados utilizada. A figura 20 demonstra a tela pela qual se configura a *ConnectionString*.

**Figura 20 - Configuração da Conexão ADO**



Na guia Provedor, deve ser informado qual o provedor de dados. No caso do Tutor Delphi, o provedor é um *driver* ODBC, pois para utilizar o banco de dados SQLAnywhere é preciso criar um *driver* ODBC. Na guia Conexão, identifica-se o nome da fonte de dados (neste caso o nome do *driver* ODBC), e o usuário e senha de acesso. Na guia Avançado, são configuradas as permissões etc. Na guia Todas, é demonstrado um resumo das configurações.

A conexão configurada para o Tutor Delphi, resulta em uma descrição, conforme o quadro 4.

#### Quadro 4 - String de conexão ADO

```
Provider=MSDASQL.1;Password=sql;Persist Security Info=True;User  
ID=dba;Data Source=SQLANY;Mode=Read|Write;Connect  
Timeout=120;Extended Properties="DSN=SQLANY;UID=dba;PWD=sql";Locale  
Identifier=1033
```

Como o Tutor Delphi é distribuído, são utilizados componentes *TDataSetProvider* para prover os dados de uma tabela para o cliente, permitindo, ao mesmo, a alteração, inclusão e exclusão de dados que estão na camada de dados.

Para configurar o *TDataSetProvider*, basta informar qual é a fonte de dados associada a ela, através da propriedade *DataSet*. Por exemplo, o componente DSPUsuario, da figura 19, tem a propriedade *DataSet* ligado ao componente ADOTableUsuario.

O cliente recebe um pacote de dados transmitido pelo *DataSetProvider*, utiliza este pacote como se fosse uma aplicação local, e depois aplica as alterações na base de dados.

As aplicações clientes podem acessar o *DataSetProvider* através da interface *IAppServerTCC*.

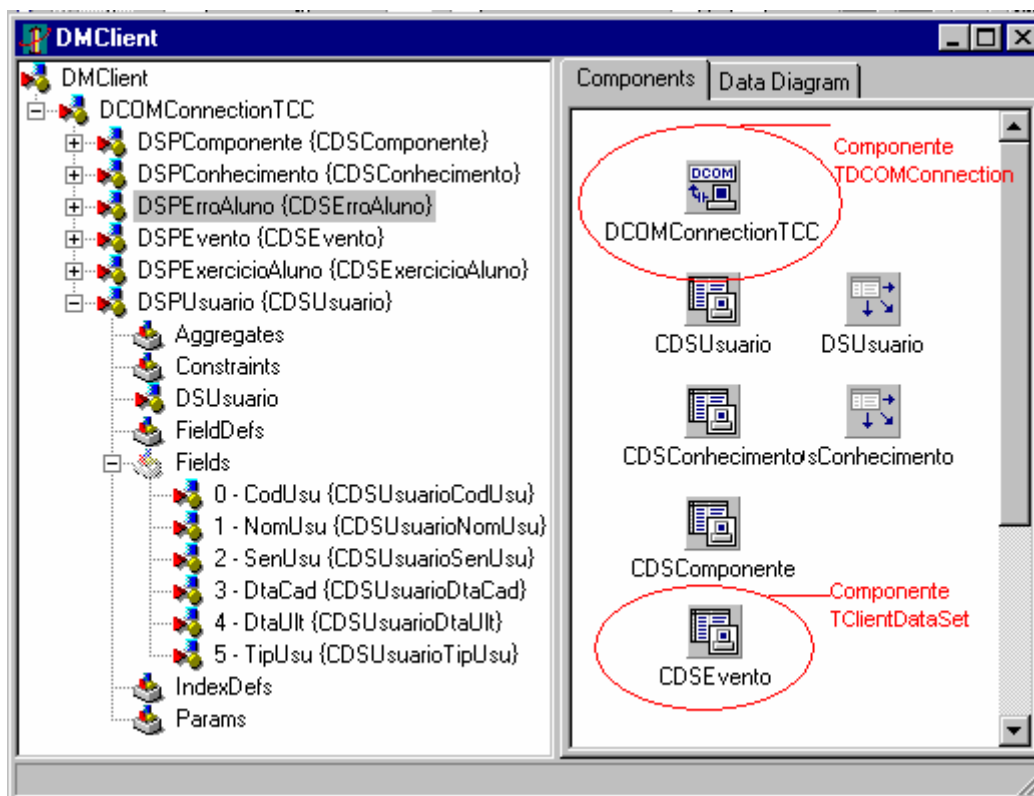
### 4.4.3 A CAMADA DE APLICAÇÃO

No lado do cliente, a programação é normal, utilizando-se as técnicas de programação com as quais o programador está acostumado a trabalhar, porém todo o acesso aos dados é feito utilizando a tecnologia ADO.

A independência de fonte de dados que o ADO permite, possibilita que o cliente execute ora em um banco de dados, ora em outro. Basta, para, isso que seja alterada a conexão do objeto *TADOConnection*.

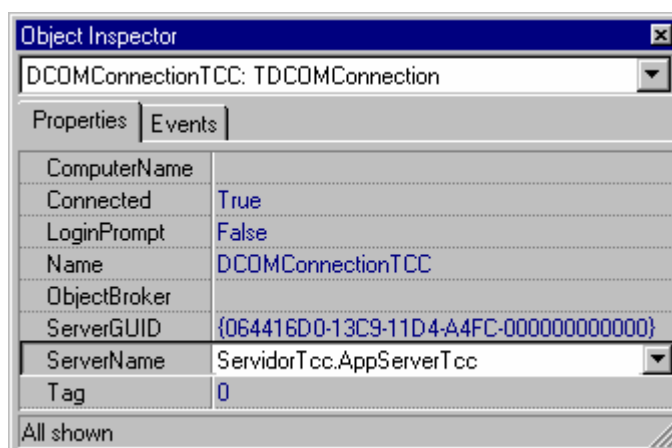
Foi criado no aplicativo cliente do Tutor Delphi, um módulo de dados simples. A figura 21 demonstra este módulo de dados.

**Figura 21 - Módulo de Dados Cliente**



Para fazer a conexão com o ServidorTCC, utilizou-se o componente *TDCOMConnection*, que pode ser visualizado na figura 21.

**Figura 22 - Propriedades do DCOMConnection**



Conforme demonstra a figura 22, existem algumas propriedades que devem ser configuradas para se estabelecer uma conexão DCOM no Delphi.

- a) a propriedade *ComputerName* deve conter o nome do computador que irá servir como servidor. Se estiver em branco, é a máquina local;
- b) a propriedade *Connected*, identifica se a conexão deve ser feita em tempo de projeto;
- c) a propriedade *LoginPrompt*, identifica se deve ser pedido um login de acesso antes de conectar ao banco de dados;
- d) a propriedade *ServerName*, identifica o servidor DCOM que deverá ser conectado. Ao informar o *ServerName*, automaticamente será preenchida a propriedade *ServerGUID*.

O acesso aos dados no cliente é feito através de componentes *TClientDataSet*. Estes componentes, através da interface *IAppServerTCC*, comunicam-se com os *DataSetProviders* do módulo de dados remoto, identificado pelo *TDCOMConnection*. Existem duas propriedades principais que devem ser informadas no *TClienteDataSet*:

- a) *remoteServer*: identifica qual é o servidor remoto. Nesta propriedade é informado o nome da conexão, neste caso: *DCOMConnectionTCC*;
- b) *providerName*: Nome do provedor de dados relacionado, que está no módulo de dados remoto do servidor DCOM. Ex: O Componente CDSEvento (ver figura 21) tem como *ProviderName* o componente DSPEvento (ver figura 19) do ServidorTcc.

Para implementação do objeto cliente no Tutor Delphi, foram utilizados componentes **MIDAS** (*Middle-tier Application Services*) do Delphi 5.0. O MIDAS 3 oferece suporte para a maioria dos principais padrões de comunicação distribuída, incluindo DCOM, MTS, Soquetes TCP/IP, HTTP e CORBA. O componente *TDCOMConnection* é um componente MIDAS 3.

Para a comunicação entre os dois lados do aplicativo (cliente e servidor) é utilizada a interface *IAppServerTCC*. Esta interface tem os seguintes métodos padrões:

- a) *as\_applyUpdates*: aplica todas as alterações que foram feitas no conjunto de dados que está no cliente, gravando-as na tabela física que se encontra na camada de dados;
- b) *as\_getRecords*: retorna os registros de um conjunto de dados. Um componente *TClientDataSet*, por exemplo, utiliza este método quando é ativado;



- c) *as\_execute*: permite que o conjunto de dados cliente interaja com *queries* ou *stored procedures* remotamente. Executa um comando;
- d) *as\_dataRequest*: é uma requisição de dados do conjunto de dados;
- e) *as\_getProviderNames*: Retorna uma lista de todos os servidores compatíveis que estão em um Módulo de Dados Remoto;
- f) *as\_getParams*: retorna o valor de um parâmetro;
- g) *as\_rowRequest*: retorna informação de um registro específico.

Raramente será preciso chamar estes métodos diretamente, pois os componentes do Delphi para serem usados nos lados cliente e servidor do aplicativo incorporam estas chamadas.

A implementação destes métodos pode ser encontrada no anexo II.

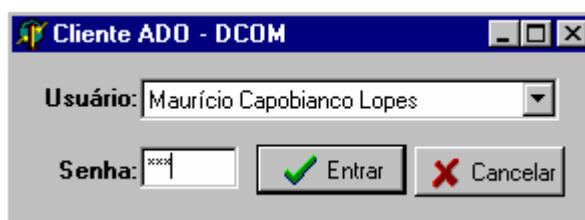
#### 4.4.4 EXECUÇÃO DE UM CASO DE USO

Para um melhor entendimento do que é executado no servidor e no banco de dados quando se faz um acesso no cliente, será demonstrado detalhadamente a execução de um dos casos de uso do sistema. Para esta demonstração, foi escolhido o caso de uso Cadastrar Usuário.

Ao iniciar a execução do cliente, o servidor é automaticamente iniciado, caso não haja ainda um servidor executando. Se houver, não será iniciado outro servidor.

A primeira tela que aparece para o usuário no cliente, é a tela de acesso, na qual o usuário escolhe o seu nome, pois parte-se da premissa de que o professor já tenha cadastrado o usuário (a não ser o primeiro usuário do sistema, que é o professor/ administrador do sistema). Logo após, informa a sua senha, conforme a figura 23.

**Figura 23 - Tela de acesso do usuário**



No campo “Usuário” irão aparecer todos os usuários cadastrados até o momento. O cliente irá buscar estas informações fazendo uma requisição para a camada intermediária, através de um componente *TClientDataSet*, que está ligado ao seu respectivo *TDataSetProvider*, que por sua vez, está enviando para o cliente as informações oriundas da fonte *TADOTable*. Esta, ainda, está buscando as informações na camada de dados e as transmitindo para a camada da aplicação.

Ao clicar no botão “Entrar” é chamado o método “Login” da classe *TAppServerTcc*, passando como parâmetro o código do usuário que quer ter o acesso. Este método é responsável por validar a senha e emitir uma mensagem de erro, caso a senha esteja incorreta, impedindo o acesso ao programa.

A classe “Usuario” é criada no momento em que a senha é validada pelo método “Login”. As propriedades desta classe, como o nome do usuário, seu código e seu tipo, são utilizadas ao longo da execução do Tutor Delphi.

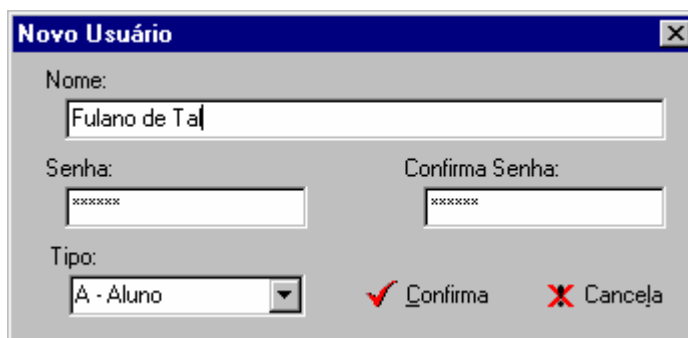
Se o acesso for concedido ao usuário, é iniciado o Tutor Delphi.

**Figura 24- Tela inicial do Tutor Delphi**



O professor, então, solicita a criação de um novo usuário, através do botão “Novo Usuário”, e informa os dados do novo usuário.

**Figura 25 - Entrada de novo usuário**



Ao confirmar a entrada do usuário, são gravados os dados na classe usuário, e a senha é criptografada antes de ser gravada. No momento em que é gravada a propriedade “SenhaUsu” da classe *TAppServerTcc*, é feita a criptografia (ver código fonte no anexo II).

É importante salientar que, quando o cliente está interagindo sobre os dados, ele na verdade não está agindo diretamente sobre os dados que estão no banco, mas sim, sobre um "espelho" dos mesmos, que é trazido do servidor, através da conexão DCOM. Este espelho é chamado "delta". Quando são aplicadas as alterações feitas no cliente, para o banco de dados, é feita uma comparação entre as diferenças do delta com as informações que o banco de dados possui naquele momento, e então as alterações são aplicadas. Para aplicar efetivamente as alterações/inclusões feitas na aplicação cliente, referentes à classe usuário, é utilizado o método *ApplyUpdates*, da interface *IAppServerTcc*. O *ApplyUpdates* faz uma a comparação entre os dados da base com os dados manipulados na aplicação (delta). As diferenças, sejam alterações, exclusões ou inclusões de informações, são reportadas à camada de dados e lá gravadas. Por não ser foco principal do trabalho, a questão multi-usuário não foi tratada, mas certamente este é um ponto que deve ser melhorado, para garantir a integridade dos dados.

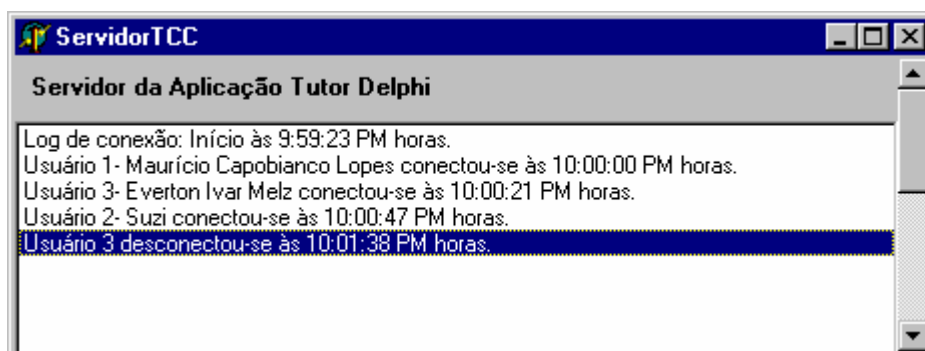
Assim também ocorre nos demais casos de uso. A aplicação faz uma requisição dos dados, trabalha-os normalmente, interagindo com o usuário através da interface. Depois, é solicitada a gravação das diferenças e os dados são fisicamente alterados no banco de dados.

O funcionamento dos demais casos de uso do protótipo do Tutor Delphi é idêntico à versão anterior, com a diferença de que executa através de objetos distribuídos, utilizando a tecnologia ADO no acesso aos dados, e estes encontram-se na camada de dados.

Ao finalizar o programa, é liberada a classe “Usuario” e então é chamado um novo método da Classe *TAppServerTcc*, chamado “Desconecta”. Este método é responsável pela atualização do log no servidor da aplicação.

Através da figura 26, verifica-se que podem existir vários clientes executando simultaneamente, cada qual com um usuário diferente conectado.

**Figura 26 - Log de acesso ao ServidorTcc**



Analisando-se o log gerado, verifica-se que entre dez horas em ponto e dez horas e um minuto, havia 3 clientes executando. Depois disso, o usuário 3 desconectou-se e permaneceram 2 usuários conectados. Cada usuário, corresponde a uma aplicação cliente.

## 5 CONSIDERAÇÕES FINAIS

### 5.1 CONCLUSÕES

O poder da tecnologia de orientação a objetos está transformando a indústria de informática como é conhecida. A sua fusão com a tecnologia cliente/servidor e a Internet, está criando novas e emergentes áreas de produção, como o comércio eletrônico, por exemplo.

Além disso, muitos avanços estão ocorrendo nesta área de orientação a objetos, onde estão sendo criados novos significados para a computação distribuída, que é outra área que está crescendo muito, em virtude das necessidades do mercado. As organizações estão cada vez mais se expandindo e necessitando de informações em qualquer hora e em qualquer lugar, não importando onde seja a fonte desta informação.

Aí entra o conceito de UDA, que embora ainda esteja mais na teoria do que na prática, é um paradigma que vem conquistando desenvolvedores e usuários, devido às vantagens que o acesso universal pode oferecer.

Unindo estas evidências com o que a tecnologia ADO pode proporcionar, certamente tem-se um conjunto que promete ser muito promissor nas aplicações como um todo, seja em nível comercial ou em nível de aprendizagem.

O estudo da tecnologia ADO mostrou-se muito interessante para o aprimoramento das técnicas de acesso aos dados, uma vez que a tecnologia é realmente fácil de utilizar, conforme sua própria documentação prega, e o resultado obtido no protótipo satisfaz as expectativas.

O principal objetivo do trabalho, que era implementar o uso da tecnologia ADO no protótipo do Tutorial Inteligente para Delphi, foi atingido plenamente e os benefícios que o tutorial obteve foram o avanço tecnológico com relação ao acesso aos dados, e também, sua nova forma distribuída, que permite aos usuários o acesso mais amplo e rápido.

Referente aos objetivos secundários, que eram demonstrar a tecnologia ADO e possibilitar o funcionamento do Tutorial Inteligente para Delphi, com objetos distribuídos ou pela Web, a conclusão é de que a demonstração do que o ADO pode proporcionar salienta-se no próprio protótipo do Tutorial Delphi. Embora este tenha sido elaborado sobre uma fonte

de dados única, oriunda do banco de dados SQL Anywhere, ele está pronto para ser utilizado com outras fontes, sem qualquer alteração na programação. O único ponto a ser alterado é a conexão ADO, apenas informando qual seria a outra fonte de dados. E em relação aos objetos distribuídos, obteve-se sucesso na implementação deste conceito, o que agrega ao Tutor Delphi uma característica inovadora e atraente, e o que é mais importante, a utilidade desta implementação.

Quanto ao item WEB, não houve tempo hábil para aprofundar os conhecimentos a este respeito e para disponibilizar o Tutor Delphi com esta característica, mas convém salientar que ainda há muita tecnologia nesta área que pode ser assimilada e aplicada não só no Tutor Delphi, mas em qualquer aplicativo, pois o acesso universal aos dados está se tornando não mais uma evidência e sim uma exigência, e um ponto importante nisso tudo, é que o ADO atende a essas exigências.

## 5.2 DIFICULDADES ENCONTRADAS

As maiores dificuldades encontradas na fase do levantamento bibliográfico foram a escassez de material, pois a tecnologia é nova e há poucos documentos sobre o assunto. É importante salientar, porém, o esforço do orientador deste trabalho no auxílio quanto às recomendações de aquisição de material informativo sobre o assunto.

Outro fator negativo, é o fato da tecnologia ser originária e proprietária da *Microsoft*®, e a maior parte do material encontrado ser da própria empresa. Teve que se tomar muito cuidado com a “propaganda” referente ao assunto.

Quanto à implementação, foram encontradas pequenas dificuldades também oriundas da novidade. Tecnologia nova em ambiente de programação novo, mas tudo contribuiu para o aprendizado e para as avaliações conclusivas.

### 5.3 SUGESTÕES

Uma sugestão é possibilitar o acesso ao Tutor Delphi pela WEB. Para isso, poderia ser aprofundar os estudos de *ActiveX* e ASP.

Sugere-se que sejam feitos tratamentos mais rigorosos quanto à segurança do sistema, pois ainda está muito vulnerável. Poderia ser feito um estudo sobre criptografia e ataques de *hackers*, controle multi-usuário, implementando técnicas de segurança no Tutor Delphi.

Outra preocupação, é com relação a independência de plataforma. Não foram realizados testes em outras plataformas, mas a teoria evidencia que como a aplicação é baseada em objetos e nas tecnologias COM e OLE, poderia haver camadas em plataformas diferentes. A sugestão é de que sejam feitos aplicativos baseados em COM ou DCOM, usando ou não ADO, e verificar a viabilidade destes em plataformas diferentes do Windows, ou interagindo com o Windows e outras plataformas simultaneamente.

# **ANEXO I**



Neste anexo encontra-se o código fonte do Módulo de Dados Remoto e o código fonte da Type Library do ServidorTCC

### a) código fonte do módulo de dados remoto:

```

unit RDMServ;

interface

uses
  Windows, Messages, SysUtils, Classes, ComServ, ComObj, VCLCom, DataBkr,
  DBClient, ServidorTcc_TLB, StdVcl, Db, ADODB, Provider, DBTables, UfServ;

type
  TAppServerTcc = class(TRemoteDataModule, IAppServerTcc)
    ADOTableUsuario: TADOTable;
    ADOTableConhecimento: TADOTable;
    ADOTableUsuarioCodUsu: TIntegerField;
    ADOTableUsuarioNomUsu: TStringField;
    ADOTableUsuarioSenUsu: TStringField;
    ADOTableUsuarioDtaCad: TDateTimeField;
    ADOTableUsuarioDtaUlt: TDateTimeField;
    ADOTableUsuarioTipUsu: TStringField;
    DSPComponente: TDataSetProvider;
    DSPEvento: TDataSetProvider;
    DSPExercicioAluno: TDataSetProvider;
    ADOTableComponente: TADOTable;
    ADOTableEvento: TADOTable;
    ADOTableExercicioAluno: TADOTable;
    ADOTableComponenteCodExe: TIntegerField;
    ADOTableComponenteCodCom: TStringField;
    ADOTableComponenteQtdCom: TIntegerField;
    ADOTableEventoCodExe: TIntegerField;
    ADOTableEventoCodCom: TStringField;
    ADOTableEventoOrdCri: TStringField;
    ADOTableEventoCodEve: TStringField;
    ADOTableEventoDesEve: TStringField;
    ADOTableEventoBinEve: TStringField;
    DSPUsuario: TDataSetProvider;
    DSPConhecimento: TDataSetProvider;
    ADOConnection1: TADOConnection;
    ADOTableConhecimentoCodExe: TIntegerField;
    ADOTableConhecimentoNomExe: TStringField;
    ADOTableConhecimentoNomAut: TStringField;
    ADOTableConhecimentoDesExe: TStringField;
    ADOTableConhecimentoTotCom: TIntegerField;
    ADOTableConhecimentoImagem: TStringField;
    ADOTableExercicioAlunoCodUsu: TIntegerField;
    ADOTableExercicioAlunoCodExe: TIntegerField;
    ADOTableExercicioAlunoCorrigido: TIntegerField;
    ADOTableExercicioAlunoVezeCorrigido: TIntegerField;
    ADOTableExercicioAlunoSucessoCorrecao: TIntegerField;
    ADOTableExercicioAlunoVezeAprendido: TIntegerField;
    ADOTableExercicioAlunoSucessoAprendizado: TIntegerField;
    ADOTableExercicioAlunoErroCometido: TIntegerField;
    ADOTableExercicioAlunoClasseErro: TIntegerField;
    ADOTableExercicioAlunoSubClasseErro: TIntegerField;
    ADOTableExercicioAlunoVezeErro: TIntegerField;
  private
    Senha : WideString;
  protected
    class procedure UpdateRegistry(Register: Boolean; const ClassID, ProgID: string);
  override;
    procedure Login(Codigo: Integer; const Senha: WideString); safecall;
    function Get_SenhaUsu: WideString; safecall;
    procedure Set_SenhaUsu(const Value: WideString); safecall;
    procedure Desconecta(CodUsu: Integer); safecall;
  public
  end;

```

```

implementation

{$R *.DFM}

//Esta Procedure é criada automaticamente pelo Delphi
//Ela atualiza o registro para refletir o status do servidor COM atual
class procedure TAppServerTcc.UpdateRegistry(Register: Boolean; const ClassID, ProgID:
string);
begin
  if Register then
  begin
    inherited UpdateRegistry(Register, ClassID, ProgID);
    EnableSocketTransport(ClassID);
    EnableWebTransport(ClassID);
  end else
  begin
    DisableSocketTransport(ClassID);
    DisableWebTransport(ClassID);
    inherited UpdateRegistry(Register, ClassID, ProgID);
  end;
end;

//Valida o Login do usuário e atualiza Log de acessos
procedure TAppServerTcc.Login(Codigo: Integer; const Senha: WideString);
var
  nomusu : String;
begin
  if (AdoTableUsuario.Locate('CodUsu',Codigo,[loPartialKey])) then
  Begin
    if (AdoTableUsuario.FieldByName('SenUsu').AsString <> Senha) then
      raise Exception.create('senha Incorreta');
    NomUsu := AdoTableUsuario.FieldByName('NomUsu').AsString;
    UfServ.FServidor.ListBoxLogs.Items.Add('Usuário ' + IntToStr(codigo) + '- ' +
      NomUsu + ' conectou-se às ' + TimeToStr(time) + ' horas.');
```

```

    end;
  end;

//Atualiza Log de acessos
procedure TAppServerTcc.Desconecta(CodUsu: Integer);
begin
  UfServ.FServidor.ListBoxLogs.Items.Add('Usuário ' + IntToStr(CodUsu) + ' desconectou-se às ' +
  TimeToStr(time) + ' horas.');
```

```

  end;

//Informa a senha do usuário conectado
function TAppServerTcc.Get_SenhaUsu: WideString;
begin
  result := Senha;
end;

// Faz a criptografia da senha, antes de gravar novo usuário
procedure TAppServerTcc.Set_SenhaUsu(const Value: WideString);
var
  Apoio : String;
  Fator : Integer;
  Indice : Byte;
begin
  Apoio := value;
  Fator := 5;
  for Indice := 1 to Length(value) do
  begin
    Fator := Fator * -1;
    Apoio[Indice] := Chr(Ord(value[Indice]) + Fator);
  end;
  Senha := Apoio;
end;

initialization
TComponentFactory.Create(ComServer, TAppServerTcc,
  Class_AppServerTcc, ciMultiInstance, tmApartment);
end.

```

## b) código fonte da *Type Library*:

```

unit ServidorTcc_TLB;

{$STYPEDADDRESS OFF} // Unit must be compiled without type-checked pointers.
interface
uses Windows, ActiveX, Classes, Graphics, OleServer, OleCtrls, StdVCL, MIDAS;
const
  // TypeLibrary Versões
  ServidorTccMajorVersion = 1;
  ServidorTccMinorVersion = 0;

  LIBID_ServidorTcc: TGUID = '{064416CD-13C9-11D4-A4FC-000000000000}';

  IID_IAppServerTcc: TGUID = '{064416CE-13C9-11D4-A4FC-000000000000}';
  CLASS_AppServerTcc: TGUID = '{064416D0-13C9-11D4-A4FC-000000000000}';
Type // Tipos definidos na TypeLibrary
IAppServerTcc = interface;
IAppServerTccDisp = dispinterface;

// Declaração da Interface
AppServerTcc = IAppServerTcc;

IAppServerTcc = interface(IAppServer)
  ['{064416CE-13C9-11D4-A4FC-000000000000}']
  procedure Login(Codigo: Integer; const Senha: WideString); safecall;
  function Get_SenhaUsu: WideString; safecall;
  procedure Set_SenhaUsu(const Value: WideString); safecall;
  procedure Desconecta(CodUsu: Integer); safecall;
  property SenhaUsu: WideString read Get_SenhaUsu write Set_SenhaUsu;
end;

IAppServerTccDisp = dispinterface
  ['{064416CE-13C9-11D4-A4FC-000000000000}']
  procedure Login(Codigo: Integer; const Senha: WideString); dispid 1;
  property SenhaUsu: WideString dispid 3;
  procedure Desconecta(CodUsu: Integer); dispid 2;
  function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant;
    MaxErrors: Integer; out ErrorCount: Integer; var OwnerData:
OleVariant): OleVariant; dispid 20000000;
  function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut:
Integer;

    Options: Integer; const CommandText: WideString;
    var Params: OleVariant; var OwnerData: OleVariant): OleVariant;
dispid 20000001;
  function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
dispid 20000002;
  function AS_GetProviderNames: OleVariant; dispid 20000003;
  function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant):
OleVariant; dispid 20000004;
  function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType:
Integer;

    var OwnerData: OleVariant): OleVariant; dispid 20000005;
  procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString;
    var Params: OleVariant; var OwnerData: OleVariant); dispid 20000006;
end;

CoAppServerTcc = class
  class function Create: IAppServerTcc;
  class function CreateRemote(const MachineName: string): IAppServerTcc;
end;

{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
TAppServerTccProperties= class;
{$ENDIF}
TAppServerTcc = class(TOleServer)
private
  FIntf: IAppServerTcc;
{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
  FProps: TAppServerTccProperties;
  function GetServerProperties: TAppServerTccProperties;
{$ENDIF}

```

```

    function      GetDefaultInterface: IAppServerTcc;
protected
    procedure InitServerData; override;
    function Get_SenhaUsu: WideString;
    procedure Set_SenhaUsu(const Value: WideString);
public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure Connect; override;
    procedure ConnectTo(svrIntf: IAppServerTcc);
    procedure Disconnect; override;
    function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant;
                            MaxErrors: Integer; out ErrorCount: Integer; var OwnerData:
OleVariant): OleVariant;
    function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut:
Integer;
                            Options: Integer; const CommandText: WideString;
                            var Params: OleVariant; var OwnerData: OleVariant): OleVariant;
    function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
    function AS_GetProviderNames: OleVariant;
    function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant):
OleVariant;
    function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType:
Integer;
                            var OwnerData: OleVariant): OleVariant;
    procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString;
                        var Params: OleVariant; var OwnerData: OleVariant);
    procedure Login(Codigo: Integer; const Senha: WideString);
    procedure Desconecta(CodUsu: Integer);
    property DefaultInterface: IAppServerTcc read GetDefaultInterface;
    property SenhaUsu: WideString read Get_SenhaUsu write Set_SenhaUsu;
published
{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
    property Server: TAppServerTccProperties read GetServerProperties;
{$ENDIF}
end;

{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}

TAppServerTccProperties = class(TPersistent)
private
    FServer: TAppServerTcc;
    function GetDefaultInterface: IAppServerTcc;
    constructor Create(AServer: TAppServerTcc);
protected
    function Get_SenhaUsu: WideString;
    procedure Set_SenhaUsu(const Value: WideString);
public
    property DefaultInterface: IAppServerTcc read GetDefaultInterface;
published
    property SenhaUsu: WideString read Get_SenhaUsu write Set_SenhaUsu;
end;
{$ENDIF}

procedure Register;

implementation

uses ComObj;

class function CoAppServerTcc.Create: IAppServerTcc;
begin
    Result := CreateComObject(CLASS_AppServerTcc) as IAppServerTcc;
end;

class function CoAppServerTcc.CreateRemote(const MachineName: string): IAppServerTcc;
begin
    Result := CreateRemoteComObject(MachineName, CLASS_AppServerTcc) as IAppServerTcc;
end;

procedure TAppServerTcc.InitServerData;
const
    CServerData: TServerData = (

```

```

    ClassID:   '{064416D0-13C9-11D4-A4FC-000000000000}';
    IntfIID:   '{064416CE-13C9-11D4-A4FC-000000000000}';
    EventIID:  '';
    LicenseKey: nil;
    Version: 500);
begin
    ServerData := @CServerData;
end;

procedure TAppServerTcc.Connect;
var
    punk: IUnknown;
begin
    if FIntf = nil then
    begin
        punk := GetServer;
        FIntf := punk as IAppServerTcc;
    end;
end;

procedure TAppServerTcc.ConnectTo(svrIntf: IAppServerTcc);
begin
    Disconnect;
    FIntf := svrIntf;
end;

procedure TAppServerTcc.DisConnect;
begin
    if Fintf <> nil then
    begin
        FIntf := nil;
    end;
end;

function TAppServerTcc.GetDefaultInterface: IAppServerTcc;
begin
    if FIntf = nil then
        Connect;
    Assert(FIntf <> nil, 'DefaultInterface is NULL. Component is not connected to Server. You
must call ''Connect'' or ''ConnectTo'' before this operation');
    Result := FIntf;
end;

constructor TAppServerTcc.Create(AOwner: TComponent);
begin
    inherited Create(AOwner);
    {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
        FProps := TAppServerTccProperties.Create(Self);
    {$ENDIF}
end;

destructor TAppServerTcc.Destroy;
begin
    {$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
        FProps.Free;
    {$ENDIF}
    inherited Destroy;
end;

{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
function TAppServerTcc.GetServerProperties: TAppServerTccProperties;
begin
    Result := FProps;
end;
{$ENDIF}

function TAppServerTcc.Get_SenhaUsu: WideString;
begin
    Result := DefaultInterface.Get_SenhaUsu;
end;

procedure TAppServerTcc.Set_SenhaUsu(const Value: WideString);
begin
    DefaultInterface.Set_SenhaUsu(Value);
end;

```

```

end;

function TAppServerTcc.AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant;
                                       MaxErrors: Integer; out ErrorCount: Integer;
                                       var OwnerData: OleVariant): OleVariant;

begin
  Result := DefaultInterface.AS_ApplyUpdates(ProviderName, Delta, MaxErrors, ErrorCount,
  OwnerData);
end;

function TAppServerTcc.AS_GetRecords(const ProviderName: WideString; Count: Integer;
                                       out RecsOut: Integer; Options: Integer;
                                       const CommandText: WideString; var Params: OleVariant;
                                       var OwnerData: OleVariant): OleVariant;

begin
  Result := DefaultInterface.AS_GetRecords(ProviderName, Count, RecsOut, Options, CommandText,
  Params, OwnerData);
end;

function TAppServerTcc.AS_DataRequest(const ProviderName: WideString; Data: OleVariant):
OleVariant;
begin
  Result := DefaultInterface.AS_DataRequest(ProviderName, Data);
end;

function TAppServerTcc.AS_GetProviderNames: OleVariant;
begin
  Result := DefaultInterface.AS_GetProviderNames;
end;

function TAppServerTcc.AS_GetParams(const ProviderName: WideString; var OwnerData:
OleVariant): OleVariant;
begin
  Result := DefaultInterface.AS_GetParams(ProviderName, OwnerData);
end;

function TAppServerTcc.AS_RowRequest(const ProviderName: WideString; Row: OleVariant;
                                       RequestType: Integer; var OwnerData: OleVariant):
OleVariant;
begin
  Result := DefaultInterface.AS_RowRequest(ProviderName, Row, RequestType, OwnerData);
end;

procedure TAppServerTcc.AS_Execute(const ProviderName: WideString; const CommandText:
WideString;
                                       var Params: OleVariant; var OwnerData: OleVariant);

begin
  DefaultInterface.AS_Execute(ProviderName, CommandText, Params, OwnerData);
end;

procedure TAppServerTcc.Login(Codigo: Integer; const Senha: WideString);
begin
  DefaultInterface.Login(Codigo, Senha);
end;

procedure TAppServerTcc.Desconecta(CodUsu: Integer);
begin
  DefaultInterface.Desconecta(CodUsu);
end;

{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
constructor TAppServerTccProperties.Create(AServer: TAppServerTcc);
begin
  inherited Create;
  FServer := AServer;
end;

function TAppServerTccProperties.GetDefaultInterface: IAppServerTcc;
begin
  Result := FServer.DefaultInterface;
end;

function TAppServerTccProperties.Get_SenhaUsu: WideString;
begin

```

```
    Result := DefaultInterface.Get_SenhaUsu;
end;

procedure TAppServerTccProperties.Set_SenhaUsu(const Value: WideString);
begin
    DefaultInterface.Set_SenhaUsu(Value);
end;

{$ENDIF}

procedure Register;
begin
    RegisterComponents('Servers', [TAppServerTcc]);
end;

end.
```

## GLOSSÁRIO

- CGI – É um protocolo padrão para comunicação entre um navegador-cliente e um servidor WEB. Este protocolo permite que o navegador solicite e envie dados e tem por base a entrada e a saída de linha de comando padrão de um aplicativo.
- CICS – É um programa de processamento de transações on-line da IBM que, junto com o COBOL, formou durante as últimas décadas o conjunto mais comum de ferramentas, por construir aplicações de transação cliente no mundo de computação dos mainframes. Um grande número das aplicações ainda usa aplicações COBOL/CICS. Usando as interfaces de programação providas por CICS, um provedor pode escrever programas que se comunicam com usuários on-line interagindo com banco de dados.
- DBMS – É um sistema gerenciador de banco de dados. Um tipo de DBMS é o RDBMS, para bancos de dados relacionais. Um novo tipo de DBMS é o OODBMS, para bancos de dados orientados a objetos.
- FTP – É um serviço de transferência de arquivos, com um protocolo padrão para Internet. É o caminho mais simples para se mandar informações pela Internet.
- HTTP – É um protocolo a nível de aplicação, na Internet, que serve para fazer o transporte de hipertextos. Hipertextos são os textos das páginas de um site na WEB.
- HTML – É uma “linguagem” na qual se desenvolvem as páginas que são apresentadas na WEB.
- ISAM – Um driver para especificar permissão de acesso a banco de dados com formato como o do dBASE, *Microsoft*® Excel, e Paradox. O Jet instala estes drivers ISAM quando referenciado pela aplicação. A localização destes drivers é mantida no Registro do *Microsoft*® Windows. Também faz parte de tecnologias *Microsoft*® e portanto, é mais utilizado no Visual Basic e Access.
- JET – É um sistema de gerência de dados da *Microsoft*®, utilizado mais



amplamente no Visual Basic com Access (tecnologias *Microsoft*® )

- Jscript – É uma linguagem de script para páginas HTML, da *Microsoft*® (parecida com o Java Script).
- MSMQ – Message Queue é um método por qual processos (ou instâncias de programa) podem trocar ou podem passar dados que usam uma interface para uma fila de mensagens administrada pelo sistema operacional. Uma fila de mensagem pode ser criada através de um processo e pode ser usada por processos múltiplos de leitura e/ou gravação. Por exemplo, um processo servidor pode ler e pode escrever mensagens de e para uma fila de mensagem criada para processos de cliente. O tipo de mensagem pode ser usado para associar uma mensagem com um processo cliente particular embora todas as mensagens estejam na mesma fila. A aplicação programa (ou seus processos) cria mensagem formando a fila, e envia e recebe mensagens que usam uma interface de programa de aplicação (API). O MSMQ é o servidor de *Message Queue* da *Microsoft*®.
- SQL – Linguagem estruturada que permite consultar e realizar ações sobre dados em um banco de dados.
- VBScript – É uma linguagem de script interpretada, utilizada no Visual Basic.

## REFERÊNCIAS BIBLIOGRÁFICAS

- [BOR1999] BORLAND, CORPORATION, INC. **Tecnologia ActiveX**. 1999, Endereço eletrônico : <http://www.delphibrasil.com/ActiveX.html>. Data da consulta: 25/02/2000.
- [CAL1999] CALVERT, Charlei. **Accessing database using ADO and Delphi**. 1999. Endereço eletrônico: <http://www.borland.com/techvoyage/articles/ADOBasics/ADOBasics.html>. Data da consulta: 28/02/2000.
- [CAN2000] CANTU, Marco. **Dominando o Delphi 5.0 - a bíblia**. São Paulo : Makron Books, 2000.
- [CAP1999] CAPELETTO, Johni Jeferson. **Comunicação entre objetos distribuídos utilizando a tecnologia CORBA (common object request broker architecture)**. Blumenau, 1999. TCC (Bacharelado em Ciências da Computação) Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau.
- [KAU1997] KAUFMAN, Sanders, JR., Jeff Perkins, FLEET, Dina. **Aprenda em 21 dias programação ActiveX**. Rio de Janeiro : Campus, 1997.
- [FRA1997] FRANCISCO, Henrique Rolfsen. **Orientação a objetos e computação distribuída**. Developers Magazine, p. 26-28, aug 1997).
- [FRA1999] FRANCO, Danton Cavalcanti Júnior. **Protótipo de um tutorial inteligente para o ambiente Delphi**. Blumenau, 1999. TCC (Bacharelado em Ciências da Computação) Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau.
- [FUR1998] FURLAN, José Davi. **Modelagem de objetos através da UML – the unified modeling language**. São Paulo : Makron Books, 1998.

- [HON1999] HONORATO, Paulo Francioli Junior. **Objetos distribuídos**. 1999. Endereço eletrônico : <http://www.nutilus.com.br/~francioli/>. Data da consulta: 10/03/2000.
- [MAR1999] MARTINER, William, FALINO James, HERION David. **Building distributed applications with ADO**. New York : Wiley Computer Publishing, 1999.
- [MIC2000] *Microsoft*® Corporation. **Msdn library**. 2000. Endereço Eletrônico: <http://msdn.microsoft.com>. Data da consulta: 22/04/2000.
- [SES1998] SESSIONS, Roger. **COM and DCOM: Microsoft's vision for distributed objects**. New York : Wiley Computer Publishing, 1998.